Optimization Methods and Software
Vol. 21, No. 3, June 2006, 493–512

Taylor & Francis
Taylor & Francis Group

Computational enhancements in low-rank semidefinite programming

SAMUEL BURER* and CHANGHUI CHOI

S210 Pappajohn Business Building, University of Iowa, Iowa City, IA 52242-1000, USA

(Received 12 July 2004; revised in final form 30 June 2005)

We discuss computational enhancements for the low-rank semidefinite programming algorithm, including the extension to block semidefinite programs (SDPs), 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 on a wide class of test problems. In particular, the truncated-Newton variant is able to achieve high accuracy in modest amounts of time on maximum-cut-type SDPs.

Keywords: Semidefinite programming; Low-rank matrices; Vector programming; Non-linear programming; Numerical experiments

1. Introduction

In this article, 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

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

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

*Corresponding author. Tel.: +1-319-335-0931; Fax: +1-319-335-0297; Email: samuel-burer@uiowa.edu

Optimization Methods and Software
ISSN 1055-6788 print/ISSN 1029-4937 online © 2006 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/10556780500286582

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

Second-order methods (SOMs) (especially 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 SOMs publicly available [1–6]. 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 SOMs, including (i) modified SOMs in which the Newton direction is calculated using the method of preconditioned conjugate gradients (CG) [7–13]; and (ii) first-order nonlinear programming algorithms [14–20]. As a result, a number of SDPs that were unreachable by SOMs 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 because 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 because M is dense and unstructured. Moreover, computational experience with (i) and (ii) has mostly involved SDP relaxations of combinatorial optimization problems. It is unclear whether the benefits of (i) and (ii) extend to more general classes of problems.

In this article, we address issues of robustness and accuracy in first-order algorithms for SDP. More specifically, we consider how to extend and improve the so-called 'low-rank' semidefinite programming algorithm of Burer and Monteiro [14,15].

Stated briefly, the main advantage of the low-rank algorithm in relation to SOMs is its ability to avoid working with and storing the dense matrices X and M mentioned earlier. To avoid X, the algorithm implicitly stores X in the factored form $X := RR^{\mathrm{T}}$, where the number of columns of R is small relative to n. In addition, the algorithm employs a first-order approach instead of Newton's method, which allows it to avoid dense $m \times m$ matrices similar to M.

More specifically, the motivating idea for the low-rank algorithm of Burer and Monteiro is the following result of Pataki [21] and Barvinok [22]:

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* $\bar{r}(\bar{r}+1)/2 \leq m$. *As 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* $\mathrm{rank}\,(X) \leq r$, *where* $r := \min\{\bar{r}\colon \bar{r}(\bar{r}+1)/2 \geq m\}$; *note that* $r \approx \sqrt{2m}$. *Thus*, (1) *is equivalent to the non-linear program*

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

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 [14], Burer and Monteiro propose an augmented Lagrangian algorithm [23] for solving (3) which uses the limited memory BFGS (L-BFGS) approach with strong Wolfe–Powell (WP) linesearch, and in [15], they prove the convergence of the algorithm to a global optimal solution even in the face of the non-convexity of (3). We emphasize that they only studied single-block SDPs, i.e. ones with $\ell = 1$.

Regarding computation, Burer and Monteiro anticipated two advantages of the low-rank algorithm. First, the storage requirements of the low-rank algorithm are $\mathcal{O}(nr) = \mathcal{O}(n\sqrt{m})$, whereas SOMs require $O(n^2 + m^2)$ storage. This is an immediate advantage of the low-rank algorithm on problems where n and m are large enough, so that SOMs require more than, say, the 1 GB of RAM on a typical workstation. Also, they suspected the low-rank algorithm could benefit from its first-order convergence rate, when solving problems to low or to medium accuracy. That is, one would expect the algorithm to approach the optimal solution quickly in the first iterations (but then to tail off strongly as higher accuracy is required).

In both articles [14,15], Burer and Monteiro present computational results, which confirm the anticipated advantages of the low-rank approach. The test cases are large-scale SDP relaxations of combinatorial optimization problems, including the maximum cut, minimum bisection, stable set, and quadratic assignment problems.

Using the algorithm of [14,15] 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 the 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 [23,24] 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 section 4, we discuss some progress towards solving (1) and (2) to high accuracy for certain classes of problems using the truncated-Newton method. We also propose several preconditioning strategies and explore their relative advantages and disadvantages. 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 SOMs.

In section 5, we conclude with a few final remarks, including a brief discussion of how the improved low-rank algorithm presented compares with SOMs in practice.

Throughout the article, we provide computational results on a collection of large-scale SDPs representing a number of different problem classes. Although the 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. Overall, our main conclusions are:

- the low-rank algorithm may be successfully applied to block SDPs;
- the exact linesearch signficantly improves the speed and robustness of the low-rank algorithm;
- the rank-reduction procedure can provide computational savings, though more research is needed to improve its reliability;
- the truncated-Newton approach signficantly outperforms L-BFGS on SDPs that arise as relaxations of maximum cut and related problems; on other classes of problems, L-BFGS outperforms truncated-Newton;

- the truncated-Newton approach can be used to solve maximum-cut and related SDPs to high accuracy in modest amounts of time;
- preconditioning strategies can be used to reduce the number of Hessian-vector products in the truncated-Newton approach, although, in most cases, the added computational overhead incurred by preconditioning outweighs the savings gotten from a reduction in the number of Hessian-vector products;
- on all problems where memory is a concern, the low-rank algorithm is a viable alternative to SOMs.

2. Some simple enhancements

In this section, we discuss several simple, but powerful, enhancements to the low-rank algorithm of Burer and Monteiro [14,15]. Computational results illustrating the improvements are also provided.

2.1 *Extension to block SDPs*

As mentioned in section 1, 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 '$X = RR^{\mathrm{T}}$' 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 kth block of X by X_k, for $k = 1, \ldots, \ell$ and will substitute $X_k = R_k R_k^{\mathrm{T}}$, where $R_k \in \Re^{n_k \times r_k}$. Our concern is how to choose r_k.

Pataki [21] 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 non-zeros of an extreme point by m. This can be seen by simply noting that* (1) *is an LP when $n_1 = \cdots = n_\ell = 1$. However, Theorem* 2.1, *does not immediately suggest how to generalize the low-rank algorithm to multiple blocks because 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 kth block of X.

PROPOSITION 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 $\bar{r}_k(\bar{r}_k + 1)/2 \leq m_k$.*

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

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

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 equation (1) is equivalent to

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

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 $X_k = R_k R_k^{\mathrm{T}}$. 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. For example, the remaining constraints $X_k \in \mathcal{S}_+^{n_k}$ could be handled using techniques from SOMs. (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 equation (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

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

$y \in \Re^m$ 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 section 1, Burer and Monteiro [14] 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. As $f(\alpha)$ 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 because $\mathcal{L}$ 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 [25], which simply requires the coefficients of the cubic polynomial $f'(\alpha)$. It is straightforward to verify that

$$f(\alpha) = a\,\alpha^4 + b\,\alpha^3 + c\,\alpha^2 + d\,\alpha + e,$$
$$f'(\alpha) = 4\,a\,\alpha^3 + 3\,b\,\alpha^2 + 2\,c\,\alpha + d,$$

where

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

and

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

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 $f'(\alpha)$ is 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 $X_k = R_k R_k^{\mathrm{T}}$ 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 with 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 [26], 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.* [27], which is implemented in the LAPACK library [28].) 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 with 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 $LL^{\mathrm{T}} \approx \tilde{L}\tilde{L}^{\mathrm{T}}$, which in turn implies

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

In contrast, if $s_k' > s_k$, then we update R_k to $R_k' = (R_k\ E)$, 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 $R_k'(R_k')^{\mathrm{T}}$ relatively close to $R_k R_k^{\mathrm{T}}$.

It is important to point out that, in theory, determining the numerical rank s by the procedure outlined earlier 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 $\mathrm{rank}(X_k) > s_k$. Accordingly, we consider it dangerous to keep $s_k < r_k$ if $s = s_k$ because 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 subsection 2.4, 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 subsequently 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 10th 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 $\rho_c = 10^{-3}$ because our goal then will be to achieve higher accuracy. This stopping criterion has two aspects: first, as $\nabla \mathcal{L}(R) = 2\,S\,R$ (see equation (11) in section 3), one can interpret the inequality as enforcing complementary slackness between $X := RR^{\mathrm{T}}$ and S; secondly, 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.

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

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 h (or 18,000 s) 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 $C \bullet RR^{\mathrm{T}}$, the final infeasibility $\|b - \mathcal{A}(RR^{\mathrm{T}})\|/(1 + |b_{\max}|)$ in accordance with (10), and the time (in seconds). We also report the following measure of 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:

$$\mathrm{opt}(S) := \frac{\max\{0, -\lambda_{\min}[S]\}}{1 + |C_{\max}|}.$$

The eigenvalue is calculated using the package ARPACK [29]. 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. Throughout the tables, we will

Table 1. Large-scale test problems, including: (i) maximum-cut relaxations and related, some with $m \approx n$ and some with $m \gg n$; (ii) Lovász theta instances; (iii) relaxations of polynomial optimization problems; (iv) electronic-structure instances; and (v) miscellaneous problems from various sources.

Instance	Nickname	ℓ	n	r	m	Non-zeros	Source
biomedP	biomed	1	6,514	115	6,515	21,862,222	[36]
cancer_100	cancer	1	569	145	10,469	172,634	[30]
foot	foot	1	2,208	67	2,209	4,879,684	[30]
equalG51	G51	1	1,001	45	1,001	508,411	[37]
G40_mb	G40	1	2,000	64	2,001	2,016,474	[9]
G59mc	G59	1	5,000	101	5,000	38,858	[30]
hand	hand	1	1,296	51	1,297	1,682,208	[38]
ice_2.0	ice	1	8,113	128	8,113	40,380	[39]
ODSAP20	ODSAP20	1	190	52	1,330	6,149	[39]
ODSAP30	ODSAP30	1	435	95	4,495	20,820	[39]
ODSAP40	ODSAP40	1	780	147	10,660	53,677	[30]
p_auss2_3.0	p_auss	1	9,115	136	9,115	45,381	[30]
rendl1_600_0	rendl	1	600	35	601	361,200	[30]
brock400_4.co_th	brock	1	400	201	20,078	100,677	[14]
p_hat300-1.co_th	p_hat	1	300	261	33,918	79,367	[14]
theta6	theta	1	300	94	4,375	49,824	[37]
cphil12	cphil	1	364	158	12,376	66,430	[30]
neu3	neu3	3	420	123	7,364	87,575	[30]
neu3g	neu3g	1	462	127	8,007	106,953	[30]
rabmo	rabmo	6,607	6,826	6,707	5,004	60,306	[30]
rose15	rose	3	137	89	3,860	9,184	[30]
taha1a	taha	14	1,680	787	3,002	177,429	[30]
BeH_2Sigma+_STO-6GN5r12g1T2	BeH	115	1,406	559	948	66,572	[40]
BH2_2A1_STO-6GN7r14g1T2	BH2	143	2,166	755	1,743	143,510	[40]
H3O+_1-A1_STO-6GN10r16g1T2_5	H3O	175	3,162	980	2,964	279,898	[40]
neosfbr20	neos	1	362	121	7,201	366,184	[30]
shmup5	shmup	3,602	11,041	123	1,800	119,223	[41]
swissroll	swissroll	1	800	83	3,380	331,337	[30]
yalsdp	yalsdp	3	300	300	5,051	1,005,350	[30]

collect the problems in five groups, roughly according to problem type: the first group contains maximum-cut SDP relaxations and related, some with $m \approx n$, and some with $m \gg n$; the second group contains Lovász theta instances; the third group contains SDP relaxations of polynomial optimization problems; the fourth group contains SDPs arising in electronic structure calculations; and the last group is a 'miscellaneous' collection. (For the interested reader, most of the problems are archived at H. Mittelmann's website [30].)

As discussed in [14], the low-rank algorithm can exploit low-rank structure in the data matrices C and A_i when evaluating function and gradients. For example, the Lovász theta instances have $C = -ee^{\mathrm{T}}$, where e is the all-ones vector. Instead of calculating $C \bullet RR^{\mathrm{T}}$ in the straightforward way, it is quicker to calculate $-\|R^{\mathrm{T}}e\|^2$. (It is important to keep in mind that the two uses of the term 'low-rank' are actually independent of one another; one refers to algorithm design and the other refers to features inherent in the data.) By preprocessing the data matrices using eigenvalue decompositions, one can exploit low-rank structure in the data if it is present, which is done in our implementation. Only the theta problems and a few of the maximum-cut-type SDPs have this structure.

Tables 2 and 3 give results for three variants of the L-BFGS low-rank algorithm: with strong WP linesearch (labeled WP), exact linesearch (labeled EXACT), and exact linesearch with rank reduction (labeled REDUCE). Comparing WP and EXACT, we see a general trend that EXACT

Table 2. Objective values and times (in s) for the L-BFGS low-rank algorithm with strong WP linesearch EXACT, and exact linesearch with (REDUCE). An upper bound of 18,000 s was enforced.

	Objective value			Time (s)		
Instance	WP	EXACT	REDUCE	WP	EXACT	REDUCE
biomed	33.6285	33.6005	33.5998	18,000	18,000	18,000
cancer	−27,623.9241	−27,623.9203	−27,623.8232	18,000	18,000	18,000
foot	−585,298.4170	−585,298.3470	−585,298.3480	8,506	3,525	2,347
G51	−4,004.9557	−4,005.1200	−4,005.5560	152	78	122
G40	−2,864.3209	−2,864.3222	−2,864.3227	805	570	629
G59	−14,624.5440	−14,624.4824	−14,624.4941	1,596	777	751
hand	−24,747.7792	−24,747.7791	−24,747.7794	4,889	3,967	2,724
ice	6,823.8961	6,822.4311	6,820.5022	745	599	515
ODSAP20	15,285.9444	15,285.9479	15,285.9385	214	95	104
ODSAP30	43,963.9072	43,964.2664	43,964.2645	3,756	1,386	1,365
ODSAP40	168,104.4300	168,042.2280	168,042.1080	18,000	18,000	18,000
p_auss	8,637.2269	8,623.7122	8,625.2257	1,016	1,660	1,133
rendl	−55,796.8758	−55,796.8755	−55,796.8753	279	96	275
brock	−39.7013	−39.7013	−39.7013	2,446	1,509	1,174
p_hat	−10.0680	−10.0680	−10.0680	1,725	1,113	701
theta	−63.4770	−63.4770	−63.4770	335	176	168
cphil	0.0000	0.0000	0.0000	67	36	36
neu3	0.0000	0.0000	0.0000	2,297	645	670
neu3g	0.0000	0.0000	0.0000	4,176	856	826
rabmo	−3.7272	−3.7272	−3.7272	13,571	5,340	5,609
rose	−0.0030	−0.0037	−0.0016	757	305	239
taha	−0.9994	−0.9999	−0.9999	18,000	18,000	18,000
BeH	16.6940	16.6941	16.6943	369	131	89
BH2	30.4319	30.4318	30.4317	1,966	1,270	1,560
H3O	90.1130	90.1147	90.1107	10,645	4,137	10,902
neos	2,385.6046	2,385.5939	2,385.5934	4,843	2,723	3,073
shmup	−23,031.4059	−23,044.4733	−23,044.4687	18,000	18,000	18,000
swissroll	−562,529.2910	−560,581.5430	−558,323.6290	18,000	18,000	18,000
yalsdp	−1.7921	−1.7921	−1.7921	220	99	92

takes much less time to meet the feasibility tolerance, while typically achieving slightly better accuracy in the optimality measurement. Even on those problems which did not converge in the time allotted, EXACT tends to have 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.

An additional perspective on this analysis is given in figure 1, which is a performance profile of the three methods (with regard to solution time) as proposed by Dolan and Moré [31]. Roughly speaking, each curve is a cumulative distribution function, which measures the probability that the specified variant will solve a problem within a certain factor of the fastest variant on that problem. For example, based on this test set, it is 90% likely that REDUCE will finish within a factor of 1.2 of the fastest time. This chart makes it particularly clear that both EXACT and REDUCE outperform INEXACT.

Table 3. Infeasibility and optimality (opt(S)) measurements for the L-BFGS low-rank algorithm with strong WP linesearch EXACT, and exact linesearch with REDUCE.

	Infeasibility			Optimality		
Instance	WP	EXACT	REDUCE	WP	EXACT	REDUCE
biomed	2E−3	5E−5	8E−5	3E−5	1E−9	8E−5
cancer	5E−5	5E−5	4E−5	1E−2	5E−3	7E−3
foot				1E−3	1E−4	2E−5
G51				4E−6	6E−6	5E−7
G40				3E−6	7E−7	1E−6
G59				2E−5	2E−5	2E−5
hand				2E−6	1E−9	9E−6
ice				2E−6	2E−6	2E−6
ODSAP20				2E−5	2E−5	1E−5
ODSAP30				1E−5	1E−5	5E−5
ODSAP40	1E−3	1E−4	2E−4	2E−4	8E−4	1E−4
p_auss				3E−6	4E−7	4E−6
rendl				2E−5	2E−5	1E−4
brock				1E−2	1E−2	1E−2
p_hat				1E−2	2E−2	5E−3
theta				4E−3	4E−3	4E−3
cphil				7E−4	7E−4	7E−4
neu3				5E−4	2E−4	4E−4
neu3g				3E−4	3E−3	3E−3
rabmo				1E−4	6E−5	4E−5
rose				1E−2	3E−3	5E−3
taha	2E−4	1E−4	7E−5	6E−3	6E−4	3E−3
BeH				1E−3	9E−4	5E−4
BH2				4E−3	2E−3	6E−4
H3O				6E−4	2E−3	2E−4
neos				1E−4	4E−5	3E−5
shmup	1E−1	2E−4	2E−4	9E−3	4E−4	8E−4
swissroll	2E−4	1E−4	7E−5	4E−3	3E−3	3E−3
yalsdp				2E−4	3E−4	1E−4

Note: For both measurements, smaller values are better. Infeasibilities are only shown when the algorithm did not meet the feasibility tolerance of 10^{-5} within the 18,000 seconds allotted.

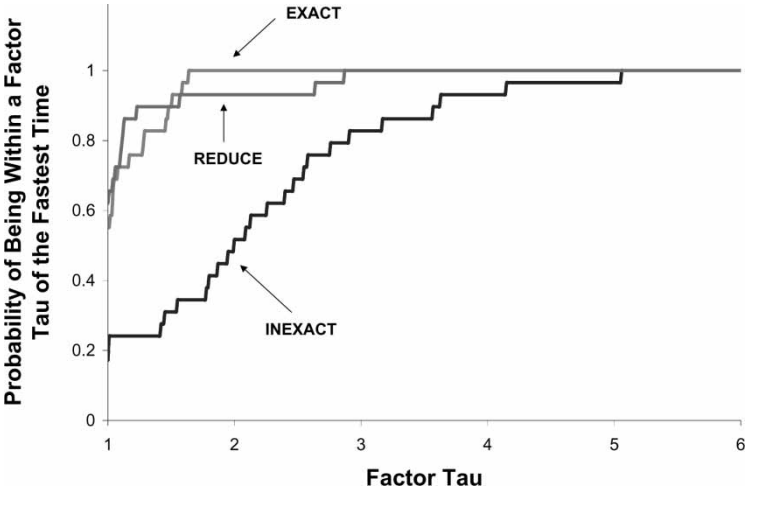

Figure 1. Performance profile of three variants of the low-rank algorithm.

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 [23,24]. In the context of non-convex minimization, the basic idea is to attempt the calculation of the Newton direction using the method of CG for positive definite systems, which generates better and better descent steps approximating the Newton direction during its execution. However, as 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 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 (hopefully) further reducing the number of Hessian-vector products are discussed in section 4.

The gradient of (7) is given by

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

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

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

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*

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

of (12) *can be expressed as*

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

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

Proof Equation (13) is established as

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

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.*

In other words, 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 [15] 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 non-zero only in those rows where A_i is non-zero, 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 earlier, 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 earlier, 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

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

3.1 *Computational results*

In tables 4 and 5, 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 except for the addition of (inner) iteration counts for each method on each instance.

One of the most noticeable features of table 4 is that the truncated-Newton variant takes considerably fewer iterations than the L-BFGS variant. This strongly indicates that the truncated-Newton is producing high-quality descent directions by exploiting second-order information. In contrast, on most groups of problems, the overall time required by the truncated-Newton is significantly more than required by L-BFGS. Clearly, the better descent directions are obtained with significant cost on these problems.

On the maximum-cut and related instances, however, the truncated-Newton is quite competitive and, in fact, outperforms L-BFGS on eight out of 13 of these problems. In other words, the truncated-Newton appears quite effective on this class of problems.

Table 4. Inner iterations, objective values, and times (in s) for the L-BFGS and the truncated-Newton variants of the low-rank algorithm.

	Iterations		Objective value		Time (s)	
Instance	L-BFGS	T-Newt	L-BFGS	T-Newt	L-BFGS	T-Newt
biomed	1,627	130	33.5998	33.5996	18,000	18,000
cancer	114,147	1,345	−27,623.8232	−27,623.5724	18,000	1,573
foot	1,818	246	−585,298.3480	−585,298.3330	2,347	825
G51	2,426	702	−4,005.5560	−4,005.5978	122	389
G40	2,745	40	−2,864.3227	−2,864.2009	629	140
G59	685	56	−14,624.4941	−14,620.6436	751	548
hand	1,195	57	−24,747.7794	−24,747.7783	2,724	1,070
ice	653	40	6,820.5022	6,839.6852	515	224
ODSAP20	16,602	330	15,285.9385	15,286.2204	104	242
ODSAP30	12,332	288	43,964.2645	43,967.2128	1,365	2,834
ODSAP40	24,357	215	168,042.1080	185,518.6300	18,000	18,000
p_auss	1,238	53	8,625.2257	8,639.8048	1,133	376
rendl	4,742	130	−55,796.8753	−55,794.1316	275	33
brock	3,023	683	−39.7013	−39.7019	1,174	2,948
p_hat	7,742	862	−10.0680	−10.0680	701	4,955
theta	5,977	925	−63.4770	−63.4769	168	198
cphil	418	58	0.0000	0.0000	36	969
neu3	7,872	121	0.0000	0.0001	670	18,000
neu3g	8,093	122	0.0000	0.0001	826	18,000
rabmo	225,632	2,052	−3.7272	−3.7171	5,609	16,448
rose	31,497	565	−0.0016	−0.0020	239	7,068
taha	222,778	98	−0.9999	−0.9977	18,000	18,000
BeH	1,073	84	16.6943	16.6939	89	843
BH2	3,274	89	30.4317	30.4327	1,560	5,729
H3O	7,739	69	90.1107	90.1260	10,902	17,854
neos	37,884	496	2,385.5934	2,385.6248	3,073	8,573
shmup	41,008	4,715	−23,044.4687	−23,019.7134	18,000	18,000
swissroll	281,337	2,343	−558,323.6290	−556,699.9630	18,000	18,000
yalsdp	1,877	383	−1.7921	−1.7921	92	2,415

Note: Both methods use the EXACT and REDUCE techniques. An upper bound of 18,000 s was enforced.

Table 5. Infeasibility and optimality (opt(S)) measurements for the L-BFGS and the truncated-Newton variants of the low-rank algorithm.

Instance	Infeasibility		Optimality	
	L-BFGS	T-Newt	L-BFGS	T-Newt
biomed	8E−5	2E−4	8E−5	1E−9
cancer	4E−5		7E−3	8E−3
foot			2E−5	1E−9
G51			5E−7	8E−8
G40			1E−6	4E−5
G59			2E−5	4E−5
hand			9E−6	8E−7
ice			2E−6	2E−6
ODSAP20			1E−5	1E−5
ODSAP30			5E−5	7E−5
ODSAP40	2E−4	7E−2	1E−4	9E−4
p_auss			4E−6	2E−6
rendl			1E−4	3E−5
brock			1E−2	6E−4
p_hat			5E−3	7E−3
theta			4E−3	1E−9
cphil			7E−4	4E−5
neu3		2E−5	4E−4	6E−3
neu3g		3E−5	3E−3	5E−3
rabmo			4E−5	2E−4
rose			5E−3	3E−3
taha	7E−5	4E−2	3E−3	2E−1
BeH			5E−4	6E−4
BH2			6E−4	3E−4
H3O			2E−4	2E−4
neos			3E−5	8E−5
shmup	2E−4	2E−5	8E−4	3E−1
swissroll	7E−5	8E−5	3E−3	1E−3
yalsdp			1E−4	3E−1

Note: Both methods use the EXACT and REDUCE techniques. For both measurements, smaller values are better. Infeasibilities are only shown when the algorithm did not meet the feasibility tolerance of 10^{-5} within the 18,000 s allotted.

4. Preconditioning the Newton system

In this section, we discuss preconditioning strategies to (hopefully) reduce the number of Hessian-vector products required by the truncated-Newton method in each iteration of the low-rank algorithm. Our goal is to obtain higher accuracy with less overall effort. Unfortunately, we find that, although the preconditioning can be effective in reducing the number of Hessian-vector products, on most problems the gains from this reduction are 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 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, whereas the second makes use of the structures of the Hessian as described in section 3.

The first is the limited memory BFGS preconditioner of Morales and Nocedal [32]. Although 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. As 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 subsequently.

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

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

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

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

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 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*

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

where e signifies the all-ones vector of appropriate dimension, and $\mathcal{W} := \{WV \colon V \in \mathcal{V}\}$. Then, $W^* \in \mathcal{W}$ and $WW^{\mathrm{T}} \succeq \hat{W}\hat{W}^{\mathrm{T}}$ for all $\hat{W} \in \mathcal{W}$. Moreover, $W^*(W^*)^{\mathrm{T}}$ yields the best approximation of WW^{T} in the sense that W^* minimizes $\mathrm{trace}(WW^{\mathrm{T}} - \hat{W}\hat{W}^{\mathrm{T}})$ *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 $WW^{\mathrm{T}} \succeq \hat{W}\hat{W}^{\mathrm{T}}$ is equivalent to the condition $I \succeq VV^{\mathrm{T}}$, which holds if and only if $\lambda_{\max}[VV^{\mathrm{T}}] \leq 1$, which can, in turn, be shown by the Gerschgorin circle theorem and the definition of $\mathcal{V}$.

Minimizing the function $\mathrm{trace}(WW^{\mathrm{T}} - \hat{W}\hat{W}^{\mathrm{T}})$ 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. As 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 $\mathrm{trace}(WW^{\mathrm{T}} - \hat{W}\hat{W}^{\mathrm{T}})$ 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 [33,34].

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 $\hat{U} := Q\hat{D}Q^{\mathrm{T}}$ is the closest rank-$\hat{q}$ matrix to U in both the Frobenuius norm and the two-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 [29].

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 subsection 4.1 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 earlier. However, in this subsection, we would at least like to demonstrate that preconditioning can 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 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 eight of the problems from table 1 (namely, those problems for which the truncated-Newton outperformed L-BFGS in section 3.1). 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 earlier.

Table 6 gives the total number of CG iterations for all preconditioning strategies on all instances that were completed within the enforced time limit. These results, which are indicative of many tests that we performed, illustrate that the number of CG iterations can be substantially reduced in certain cases using preconditioning, but it is difficult to determine which preconditioner works best on which type of problem.

Table 6. CG iterations for eight truncated–Newton variants of the low-rank algorithm (with all variations of preconditioning, including none at all) on several maximum–cut-type problems from table 1.

	Preconditioning							
Instance	None	L-BFGS	$\hat{S}_{(d)}\hat{M}_{(z)}$	$\hat{S}_{(d)}\hat{M}_{(n)}$	$\hat{S}_{(d)}\hat{M}_{(e)}$	$\hat{S}_{(c)}\hat{M}_{(z)}$	$\hat{S}_{(c)}\hat{M}_{(n)}$	$\hat{S}_{(c)}\hat{M}_{(e)}$
cancer								
foot	**4,448**	10,732		21,354	4,615			
G40	224,635	**30,679**		156,741	242,299	317,357		
G59	2,588	3,122	5,339	6,490	**2,495**	3,669	8,911	11,023
hand	10,765	**6,931**		11,081	9,850	11,331		
ice	**2,399**	6,383		25,192	2,934	4,709	36,756	
p_auss	**2,941**	3,499		121,282	4,900	4,180		
rendl	992	42,665		6,277	**930**	33,777	39,409	38,976

Note: Bold typeface shows the smallest CG-iteration count for each instance. Here, $\rho_c = 10^{-3}$ and $\rho_f = 10^{-8}$. An upper bound of 18,000 s was enforced. CG iterations are not shown when the algorithm failed to meet the feasibility tolerance ρ_f within the 18,000 s allotted.

Table 7. Objective values, times (in s), infeasibilities, and optimalities for two truncated-Newton variants of the low-rank algorithm (one without preconditioning and one with $\hat{S}_{(d)}\hat{M}_{(e)}$-preconditioning) on the same runs displayed in table 6.

	Objective value		Time (s)		Infeasibility		Optimality	
Instance	None	$\hat{S}_{(d)}\hat{M}_{(e)}$	None	$\hat{S}_{(d)}\hat{M}_{(e)}$	None	$\hat{S}_{(d)}\hat{M}_{(e)}$	None	$\hat{S}_{(d)}\hat{M}_{(e)}$
cancer	−27623.3306	−27623.3483	18,000	18,000	1E−7	3E−7	4E−06	5E−06
foot	−585298.1600	−585298.1600	2,169	2,313			3E−12	9E−14
G40	−2864.3232	−2864.3232	5,392	17,918			1E−12	5E−13
G59	−14624.6490	−14624.6489	1,897	2,042			9E−06	9E−06
hand	−24747.7791	−24747.7791	3,640	2,767			1E−13	1E−14
ice	6809.4501	6809.2932	1,660	2,615			3E−06	3E−06
p_auss	8618.2352	8618.1246	2,119	3,827			2E−08	7E−13
rendl	−55796.8707	−55796.8707	32	30			6E−09	1E−08

Note: Infeasibilities are only shown when the algorithm did not meet the feasibility tolerance 10^{-8} within the 18,000 s allotted.

In terms of computation time, no preconditioning was the best strategy overall; the second-best strategy was $\hat{S}_{(d)}\hat{M}_{(e)}$-preconditioning. We include objective values, times, infeasibilities, and optimalities for these two strategies in table 7.

5. Final Remarks

In this article, 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, 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 have also been discussed.

A natural question that arises is the following: how does the low-rank algorithm compare to other methods, particularly SOMs? Because of the memory usage of SOMs relative to that of the low-rank approach – $\mathcal{O}(n^2 + m^2)$ versus $\mathcal{O}(n\sqrt{m})$ – we were unable to run these methods on many of our test problems because our computer has only 1 GB of RAM. In lieu of direct comparisons with SOMs, we would like to make a few comments/interpretations regarding the benchmarks of SDP software done by H. Mittelmann [35], which are publicly available on the Internet.

Mittelmann's benchmarks demonstrate that on small- to medium-scale instances, SOMs handily outperform the low-rank method (released as the package *SDPLR 1.02* and run with exact linesearch and the L-BFGS approach for computing directions) in terms of both time and accuracy. On larger instances, the situation appears to be mixed. For some problem classes, e.g. relaxations of polynomial optimization problems and instances arising in structural optimization, the low-rank method requires a very large number of iterations, and consequently, SOMs tend to outperform the low-rank algorithm. On other problems, such as electronic-structure and maximum-cut-type SDPs, the low-rank approach is quite competitive or significantly better. Finally, from Mittelmann's benchmark, one can also conclude that the low-rank approach is less susceptible to memory bottlenecks, as all but one of the SOMs in the benchmark encounter instances for which the 4 GB of RAM available in Mittelmann's computer is insufficient.

Overall, this informal comparison of the low-rank algorithm with SOMs illustrates what we believe to be the current state of SDP algorithms, namely, that the choice of algorithm should be strongly motivated by the type of SDP being solved. We are hopeful that the low-rank algorithm contributes positively to the options available to researchers and practitioners.

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 $X_k = R_k R_k^{\mathrm{T}}$ 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 article. 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 $\mathcal{A}(RR^{\mathrm{T}}) = b$ by the the inverse of the Cholesky factorization of the operator $\mathcal{A} \circ \mathcal{A}^*$, so that the resultant system $\tilde{\mathcal{A}}(RR^{\mathrm{T}}) = \tilde{b}$ would satisfy $\mathrm{cond}[\tilde{\mathcal{A}} \circ \tilde{\mathcal{A}}^*] = 1$. 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.

Acknowledgements

The authors are in debt to two anonymous referees for invaluable advice and comments that have greatly improved this article. Also, the authors would like to acknowledge the support of NSF Grant CCR-0203426.

References

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

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

[3] Kočvara, M. and Stingl, M., 2003, Pennon: a code for convex nonlinear and semidefinite programming. *Optimization Methods and Software*, **18**(3), 317–333.

[4] Sturm, J.F., 1999, Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods Software*, **11**/**12**(1–4), 625–653 (Interior point methods).

[5] Yamashita, M., Fujisawa, K., and Kojima, M., 2003, Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). *Optimization Methods and Software*, **18**(4), 491–505. The Second Japanese-Sino Optimization Meeting, Part II (Kyoto, 2002).

[6] Tütüncü, R.H., Toh, K.C. and Todd, M.J., 2001, SDPT3: a Matlab software package for semidefinite-quadratic-linear programming, version 3.0. Available online at: http://www.math.nus.edu.sg/ mattohkc/sdpt3.html.

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

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

[9] Choi, C. and Ye, Y., 2000, 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.

[10] Toh, K. and Kojima, M., 2002, Solving some large scale semidefinite programs via the conjugate residual method. *SIAM Journal on Optimization*, **12**, 669–691.
[11] Toh, K.C. 2004, Solving large scale semidefinite programs via an iterative solver on the agumented systems. *SIAM Journal on Optimization*, **14**, 670–698.
[12] Burer, S., 2003, Semidefinite programming in the space of partial positive semidefinite matrices. *SIAM Journal on Optimization*, **14**(1), 139–172.
[13] Kočvara, M. and Stingl, M., 2005, On the solution of large-scale sdp problems by the modified barrier method using iterative solvers. Research report 304, Institute of Applied Mathematics, University of Erlangen.
[14] Burer, S. and Monteiro, R., 2003, A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming (Series B)*, **95**, 329–357.
[15] Burer, S. and Monteiro, R., 2005, Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, **103**(3), 427–444.
[16] Burer, S., Monteiro, R. and Zhang, Y., 2002, Solving a class of semidefinite programs via nonlinear programming. *Mathematical Programming*, **93**, 97–122.
[17] Burer, S., Monteiro, R. and Zhang, Y., 2003, A computational study of a gradient-based log-barrier algorithm for a class of large-scale SDPs. *Mathematical Programming (Series B)*, **95**, 359–379
[18] Fukuda, M. and Kojima, M., 2000, Interior-point methods for lagrangian duals of semidefinite programs. Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguru-ku, Tokyo 152, Japan.
[19] Helmberg, C. and Kiwiel, K., 1994, A spectral bundle method with bounds. *Mathematical Programming*, **93**(2), 173–194.
[20] Helmberg, C. and Rendl, F., 2000, A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, **10**, 673–696.
[21] Pataki, G., 1998, On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of Operations Research*, **23**, 339–358.
[22] Barvinok, A., 1995, Problems of distance geometry and convex properties of quadratic maps. *Discrete Computational Geometry*, **13**, 189–202.
[23] Nocedal, J. and Wright, S., 1999, *Numerical Optimization* (New York: Springer-Verlag).
[24] Dembo, R.S., Eisenstat, S.C. and Steihaug, T., 1982, Inexact Newton methods. *SIAM Journal of Numerical Analysis*, **19**(2), 400–408.
[25] Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M. and Rossi, F. *GNU Scientific Library Reference Manual*, (2nd edn) Available online at: http://www.gnu.org/software/gsl/.
[26] Businger, P. and Golub, G.H., 1965, Handbook series linear algebra. Linear least squares solutions by householder transformations. *Numerische Mathematik*, **7**, 269–276.
[27] Quintana-Ortı, G., Sun, X. and Bischof, C.H., 1998, A BLAS-3 version of the QR factorization with column pivoting. *SIAM Journal of Scientific Computing*, 1998(5), 1486–1494 (electronic).
[28] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D., 1999, *LAPACK Users' Guide* (3rd edn) (Philadelphia, PA: Society for Industrial and Applied Mathematics).
[29] Lehoucq, R.B., Sorensen, D.C. and Yang, C., 1998, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* (Philadelphia: Society for Industrial and Applied Mathematics).
[30] Mittelmann, H.D. Available online at: http://plato.la.asu.edu/topics/testcases.html.
[31] Dolan, E.D. and Moré, J.J., 2002, Benchmarking optimization software with performance profiles. *Mathematical Programming Series A*, **91**(2), 201–213.
[32] Morales, J.L. and Nocedal, J., 2000, Automatic preconditioning by limited memory quasi-Newton updating. *SIAM Journal of Optimization*, **10**(4), 1079–1096 (electronic).
[33] Mirsky, L., 1960, Symmetric gauge functions and unitarily invariant norms. *The Quarterly Journal of Mathematics, Oxford Series 2*, **11**, 50–59.
[34] Schmidt, E., 1907, *Mathematische Annalen*, **63**, 433–476. Zur Theorie der linearen und nichtlinearen Integralgleichungen. I Tiel. Entwicklung willkürlichen Funktionen nach System vorgeschriebener.
[35] Mittelmann, H.D., 2003, An independent benchmarking of SDP and SOCP solvers. *Mathematical Programming Series B*, **95**(2), 407–430 (Computational semidefinite and second order cone programming: the state of the art).
[36] Pataki, G. and Schmieta, S., 1999, The DIMACS library of semidefinite-quadratic-linear programs. Technical report. Available online at: http://dimacs.rutgers.edu/Challenges/Seventh/Instances/.
[37] Borchers, B., 1999, SDPLIB 1.2, a library of semidefinite programming test problems. *Optimization Methods and Software*, **11**(1), 683–690.
[38] Keuchel, J., Schnörr, C., Schellewald, C. and Cremers, D., 2003, Binary partitioning, perceptual grouping, and restoration with semidefinite programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **25**(11), 1364–1379.
[39] M. Anjos, Kennings, A. and Vannelli, A., 2004, A semidefinite optimization approach for the single-row layout problem with unequal dimensions. *Discrete Optimization*, **2**(2), 113–122.
[40] Zhao, Z., Braams, B.J., Fukuda, M., Overton, M.L. and Percus, J.K., 2004, 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.
[41] Zowe, J., Kočvara and Bendsøe, M.P., 1997, Free material optimization via mathematical programming. *Mathematical Programming, Series B*, **79**(1–3), 445–466.