Improving the SDP-CUT Algorithm for Solving Semidefinite Programs

Angela Cooper

Department of Mathematics and Statistics

Southern Illinois University Edwardsville

Edwardsville, IL 62025

July 20, 2012

Abstract

In this paper, we present modifications of an interior point algorithm SDP-CUT for solving semidefinite programming (SDP) problems. This algorithm uses a cutting plane constraint in the interior and computes the weighted analytic center. Our goal is to improve the accuracy and time in solving the SDP problems. We looked into the line search methods of bisection and Nesterov-Nemirovksky's damping factor and compared them to the exact line search that had previously been used. We examined how modifying the Hessian would impact the SDP-CUT. One way of modifying the Hessian replaces the negative eigenvalues found in the sectoral decomposition. Another modification of the Hessian shifts all of its eigenvalues. We also examined the effect of a more efficient way of computing the gradient and Hessian of the barrier function. Our numerical experiments indicate that some of the modifications are effective in reducing the SDP-CUT solution time.

1 Introduction

Semidefinite programs minimize a linear function subject to a constraint, where the constraint is nonlinear, nonsmooth, but convex [2]. In semidefinite programs, the objective functions are also convex. Therefore, semidefinite programs are convex optimization problems [2].

We consider the following problem, which is referred to as a semidefinite program (SDP) [3]:

Let variable $x \in \mathbb{R}^n$, objective function $c \in \mathbb{R}^n$, and $A_i^{(j)}$ be a m_j x m_j symmetric matrix.

$$\text{minimize } c^T x$$

$$\text{subject to } A^{(j)}(x) \succeq 0, \quad j = 1, 2, \ldots, q,$$

where $A^{(j)}(x) := A_0^{(j)} + \sum_{i=1}^{n} x_i A_i^{(j)}$. $A^{(j)}(x) \succeq 0$, is called a linear matrix inequality, and $A^{(j)}(x) \succeq 0$, denotes that $A^{(j)}(x)$ is positive semidefinite [2]. We refer the the problem stated above as the primal problem.

The following are equivalent [1]:

1. $A \succeq 0$;
2. $x^T A x \geq 0 \ \forall x \in \mathbb{R}^n$;
3. $\lambda_{\min}(A) \geq 0$;
4. All principle minors of A are nonnegative;
5. $A = LL^T$ for some $L \in \mathbb{R}^{nxn}$.

We say that the primal problem is strictly feasible if $\exists x$ where $A^{(j)}(x) \succ 0$ [2].

The primal problem of the semidefinite program that was previously stated, has a dual problem associated with it [3]:

$$\text{maximize } -\sum_{j=1}^{q} A_0^{(j)} \bullet Z_j$$
$$\text{subject to } \sum_{j=1}^{q} A_i^{(j)} \bullet Z_j = c_i \text{ for } i = 1, \ldots, n$$
$$Z_j \succ 0, \text{ for } j = 1, \ldots, q.$$

The dual problem of the semidefinite program is important because it produces bounds on the optimal soultion of the primal semidefinite program. It also plays a role in the optimality conditions of the primal problem [2]. We say that the dual problem is strictly feasible if $\exists Z$ where $Z = Z^T \succ 0,\ A_i^{(j)} \bullet Z_j = c_j$ for $i = 1, \ldots, n$ [2].

We let $\mathcal{R} = \left\{\ x|\ A^{(j)}(x) \succeq 0\ \forall j = 1, \ldots, q\right\}$ be the feasible region which includes the boundary curve and the enclosed region. Let $\text{int}(\mathcal{R}) = \left\{\ x|\ A^{(j)}(x) \succ 0\ \forall j = 1, \ldots, q\right\}$ be the interior of the feasible region. Assume $\mathcal{R}$ is bounded and the $int(\mathcal{R})$ is nonempty. We also assume the set $\left\{\text{diag}(A_1^{(1)}, \ldots, A_1^{(q)}), \text{diag}(A_2^{(2)}, \ldots, A_2^{(q)}), \ldots, \text{diag}(A_n^{(1)}, \ldots, A_n^{(q)})\right\}$ is linearly independent. Let p^* denote the optimal solution of the primal problem and let d^* denote the optimal solution of the dual problem. The difference of p^* and d^* is referred to as the duality gap. We have $p^* - d^* = 0$, if the primal problem or the dual problem is strictly feasible[2].

Many interior point algorithms have been developed for solving SDPs. There are several SDP software packages available such as Sedumi and SDPT3 [7]. We studied a specific interior point method used to solve semidefinite programs called SDP-CUT [3]. This algorithm uses cutting plane constraints and weighted analytic centers. SDP-CUT had previously been implemented using Newton's method and different line search techniques. In comparison with SDPT3, SDP-CUT was found to take less iterations on many test problems.

In this paper, we present modifications of an interior point algorithm SDP-CUT for solving semidefinite programming (SDP) problems. This algorithm uses a cutting plane constraint in the interior and computes the weighted analytic center. Our goal is to improve the accuracy and time in solving the SDP problems. Our numerical experiments indicate that some of the modifications are effective in reducing the SDP-CUT solution time. We looked into the line search methods of bisection and Nesterov-Nemirovksky's damping factor and compared them to the exact line search that had previously been used. Both the bisection linesearch and the Nesterov-Nemirovsky's damping factor did not appear to make a significant improvement in the time and accuracy of the SDP-CUT. We examined how modifying the Hessian would impact the SDP-CUT. One way of modifying the Hessian replaces the negative eigenvalues found in the spectral decomposition. Another modification of the Hessian shifts all of its eigenvalues. Both modifications to the Hessian eigenvalues made a significant improvement in the running time of the SDP-CUT for most of the test problems. We also examined the effect of a more efficient way of computing the gradient and Hessian of the barrier function. The new computation of the gradient and Hessian made a significant improvement to the SDP-CUT time.

2 SDP-CUT Algorithm Description

The SDP-CUT algorithm which can be found in [3] uses cutting planes and weighted analytic centers. It iteratively corrects the feasible region until the weighted analytic center approaches the optimal solution. We will denote the current feasible region of the system of linear matrix inequalities by $\mathcal{R}_k$. The algorithm always starts with an initial feasible point, x_0^*. We will call the current feasible point x_k^*, where

$x_k^* \in \mathcal{R}_k$. This algorithm then will find a new feasible region $R_{k+1} \subset R_k$, where $\forall x \in R_{k+1}, c^T x \leq c^T x_k^*$, by adding a new constraint cut, $c^T x_k^* - c^T x + \epsilon \geq 0$. In this new constraint, ϵ guarantees that x_k^* is still a strictly feasible point in the new feasible region. The feasible point x_k^* will then move to a *weighted analytic center* of the feasible region.

Given a weight $w \in \mathcal{R}_+$, the next iterate x_{k+1}^* is the weighted analytic center

$$x_{ac} = \text{argmin}\{\phi_w(x) | x \in \mathcal{R}^n\}$$

where $\phi_w(x)$ is the weighted barrier function defined by

$$\phi_w(x) = \begin{cases} -\text{wlog}[c^T x_k^* - c^T x + \epsilon] - \sum_{j=1}^q \text{logdet}[A^{(j)}(x)] & \text{if } A^{(j)} \succ 0 \ \forall j = 1, \ldots, q \\ +\infty & \text{otherwise,} \end{cases} \tag{2}$$

The gradient $\nabla \phi_w(x)$ and the Hessian $\nabla^2 \phi_w(x)$ of the wieghted barrier function are given below for $i, \ldots, n$ and $k = 1, \ldots, n$:

$$\nabla \phi_w(x)_i = \frac{w c_i}{(c^T x_k^* - c^T x + \epsilon)} - \sum_{j=1}^q (A^{(j)}(x))^{-1} \bullet A_i^{(j)}) \tag{3}$$

$$\nabla^2 \phi_w(x)_{ik} = \frac{w c_i c_k}{(c^T x_k^* - c^T x + \epsilon)^2} - \sum_{j=1}^q [(A^{(j)}(x))^{-1} A_i^{(j)}]^T \bullet [(A^{(j)}(x))^{-1} A_k^{(j)}] \tag{4}$$

In order to find the weighted analytic center of the feasible region, Newton's

method is applied. The algorithm used to find the analytic center of a given feasible region is named WAC-NEWTON and is described below:

WAC-NEWTON

INPUT: point $y_0 \in int(\mathcal{R})$, weight vector $w \in \mathbb{R}^q$, tolerance $WTOL > 0$, and maximum number of iterations $WMAX$
Set $l = 0$
while $l < WMAX$ **do**
1. Compute the direction vector $s_l = -((\nabla^2\phi_w(y_l))^{-1}(\nabla\phi_w(y_l))$
2. Compute the Newton decrement $d=\sqrt{s_l^T(\nabla^2\phi_w(y_l))s_l}$
3. Compute stepsize α_l
4. $y_{l+1} = y_l + \alpha_l s_l$

if $d < WTOL$ **then**
break while
end if
5. $l \leftarrow l + 1$

end while
OUTPUT: $x_{ac}(w) = y_l$

In [3] different methods are used to compute the stepsize in the WAC-NEWTON algorithm. One method used a constant stepsize of $\alpha_l = 1$. Another method used called backtracking, starts with stepsize $\alpha_l = 1$. The stepsize is then reduced until a stopping condition is met. Experiments revealed that exact line search was the most efficient. To find the stepsize α_l, exact line search solves the problem below:

$$\text{minimize } \{g(\alpha) \mid \alpha > 0\},$$
$$\text{where } g(\alpha) = \phi_w(y_l + \alpha s_l)$$

One way to evaluate $g(\alpha)$ uses eigenvalues. However, the eigenvalues took too much time to compute. The method of exact line search with the following evaluation of $g(\alpha)$ was found to perform the best. This method was referred to as ELS-MAT. Here,

$$g(\alpha) = -a_l - \sum_{j=1}^{q} \text{logdet}[M_0^{(j)} + \alpha M_s^{(j)}].$$

where,

$a_l = w \log [c^T x_k^* - c^T(y_l + \alpha_l s_l) + \epsilon]$, $M_0^{(j)} = A^{(j)}(y_l)$, and $M_s^{(j)} = \sum_{i=1}^{n} (s_l)_i A_i^{(j)}$.

The SDP-CUT algorithm used to solve the semidefinite program is described below:

SDP-CUT

INPUT: $x_0^* \in \text{int}(\mathcal{R})$, weight vector $w \in \mathbb{R}_+$, $\epsilon > 0$, tolerance $STOL > 0$, and maximum number of iterations $SMAX$
Set $k = 0$
while $k < SMAX$ **do**
1. Compute cutting plan constraint $c^T x - c^T x_k^* + \epsilon \geq 0$
2. Let x_{k+1}^* be the weighted analytic center of the new system with barrier function $\phi_w(x)$ to be computed by WAC-NEWTON starting from the point x_k^*
Compute $\partial_{k+1} = c^T x_k^* - c^T x_{k+1}^*$
if $\partial_k < STOL$ **then**
break while
end if
$k \leftarrow k + 1$
end while
OUTPUT: $x_{cut}^* = x_k^*$, $p_{cut}^* = c^T x_k^*$

---

3 Improvements to the SDP-CUT Algorithm

The SDP-CUT algorithm has room for improvement. Experiments in [3] revealed that an increase in weight decreases the number of iterations. However, weights too large cause numerical errors in that as x_k^* approaches the optimal solution, the rate of convergence levels off. The choice of ϵ can also cause numerical errors. We would

ideally like a small ϵ to allow x_k^* to approach the optimal solution without getting stuck in the same position. However, too small of a choice of ϵ can cause numerical errors in that x_k^* is too close to the boundary. Larger problems create numerical errors in the Hessian because the Hessian can be no longer positive definite due to numerical errors. Also, large problems often have unbounded feasible regions. The exact line search sometimes failed or took too much time to converge.

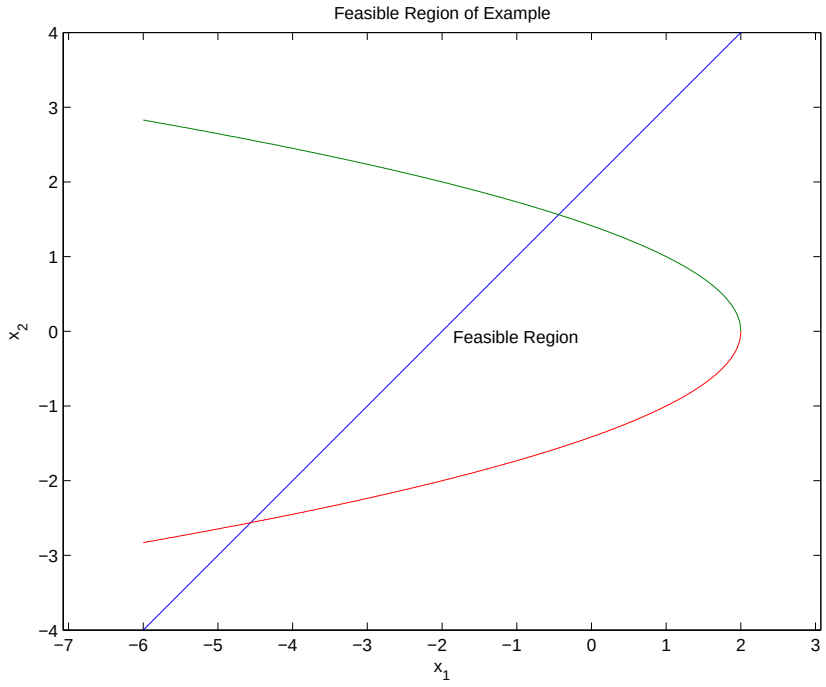
The goal is to correct these numerical errors. We would also like to investigate more efficient line search methods. The exact line search method that had previously been implemented can be very expensive. In this paper we investigate five ways to improve the SDP-CUT. Two new line search methods were implemented: the method of bisection and the Nesterov-Nemirovsky damping factor. Two different methods were used to modify the Hessian. One method replaces negative eigenvalues in the spectral decomposition of the Hessian. The other method shifts the eigenvalues in the spectral decomposition of the Hessian. The other correction method that was implemented was computing the gradient and the Hessian of the barrier function in a different manor. Each correction to the SDP-CUT was implemented individually and compared to the ELS-MAT method.

We will use the following SDP example to illustrate our techniques. In the example, $n = 2$ (number of variables), $q = 2$ (number of Linear Matrix Inequalities), $m_1 = 2$ (size of matrices in $A^{(1)}(x)$), $m_2 = 1$ (size of matrices in $A^{(2)}(x)$).

Example

$$
\begin{aligned}
\text{minimize} z &= x_1 + 2x_2 \\
\text{subject to} & \\
A^{(1)}(x) &= \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} + x_1 \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \succeq 0 \\
A^{(2)}(x) &= \begin{bmatrix} 2 \end{bmatrix} + x_1 \begin{bmatrix} 1 \end{bmatrix} + x_2 \begin{bmatrix} -1 \end{bmatrix} \succeq 0
\end{aligned}
$$

Figure 1: The feasible region of the above SDP example



3.1 Line Search Bisection

The first correction implemented was to replace the exact line search in the WAC-NEWTON algorithm used to find the step size α_l with the bisection line search. We used the bisection method to solve the problem:

$$\text{minimize } \{g(\alpha)|\alpha > 0\}$$

where, $g(\alpha) = \phi_w(y_l + \alpha s_l)$. The bisection method found in [4], starts with an interval [a,b] that is known to contain the optimal solution, α_l. The midpoint of the endpoints is calculated. Then the evaluate the convex problem at its midpoint. Then it is determined if α_l is contained the the upper or lower half of the interval. The interval is then updated to the half of the original interval that contains the optimal solution. These steps are repeated until the optimal solution is located. It should be noted that the initial interval is chosen to be [0, σ], where σ is the distance from y_l to the boundary of the feasible region in the direction of s_l.

When the bisection method is implemented, the SDP-CUT algorithm fails when the deincrement d is too small. To correct the error, if $d < \frac{1}{4}$ then the exact line search method will be used [5].

The following theorem shows how to compute the distance to the boundary of an LMI which is an eigenvalue problem.

Theorem 3.1 The distance from x_k to the boundary of the LMI $F_0 + \sum_{i=1}^{n} x_i F_i \succeq 0$ along the direction $s_k$is given by:

$$\sigma_+ = \frac{1}{\lambda_{max}(B)}$$

where $B = -F(x_k)^{-1/2}(-\sum_{i=1}^{n}(s_k)_i F_i)F(x_k)^{-1/2}$.

Proof: σ_+ is given by the optimal soltuion of the problem:

minimize σ

subject to $|F(x_k + \sigma s_k| = 0,\ \sigma > 0.$

$$\text{Now, } |F(x_k + \sigma s_k)| = 0 \quad \Leftrightarrow \quad |F(x_k) + \sigma \sum_{i=1}^{n} (s_k)_i F_i| = 0$$

$$\Leftrightarrow \quad |\frac{1}{\sigma} I + F(x_k)^{-1/2} (\sum_{i=1}^{n} (s_k)_i F_i) F(x_k)^{-1/2}| = 0$$

$$\Leftrightarrow \quad \frac{1}{\sigma} \text{ is an eigenvalue of } B = -F(x_k)^{-1/2} (\sum_{i=1}^{n} (s_k)_i F_i) F(x_k)^{-1/2}$$

So, the distance $\sigma_+ = \dfrac{1}{\lambda_{\max}(B)}$.

Line search bisection can be seen below:

Line Search Bisection

INPUT: the convex feasibility problem minimize $\{g(\alpha)|\alpha > 0\}$, the tolerance TOL, and the number of maximum iterations MAX, σ
Set $a = 0$
Set $b = \sigma$
Set $k = 0$
1. Compute the midpoint $p = \frac{a+b}{2}$
2. Evaluate $g(p)$
while $(|g(p)| > TOL)$ and $(k < MAX)$ **do**
  $k \leftarrow k + 1$
  Determine if $\alpha_l \in [a, p]$ or $\alpha_l \in [p, b]$
  if $\alpha \in [p, b]$ **then**
    $a = p$
    $b = b$
  else
    $a = a$
    $b = p$
  end if
end while
OUTPUT: α_l

3.2 Nesterov-Nemirovsky Damping Factor

This line search method calculates the stepsize α_l used in WAC-NEWTON. It applies a piece-wise function to Newton's decrement d. This method is described in [5]. The Nesterov-Nemirovsky damping factor choses the step length α_l as follows:

$$\alpha_l = \begin{cases} 1 & \text{d} \leq \frac{1}{4} \\ \frac{1}{1+d} & \text{d} > \frac{1}{4}, \end{cases} \tag{3}$$

Nesterov-Nemirovsky show that this step length allows results in $y_{l+1} = y_l + \sigma_k s_l$ is in the interior of the feasible region.

3.3 New Computation of the Gradient and Hessian

The gradient and the Hessian of the weighted barrier function $\phi_w(x)$ had previously been computed as seen in equations (6) and (7). Note that:

$$\begin{aligned} (A^{(j)}(x))^{-1}) \bullet A_i^{(j)} &= Tr((A^{(j)}(x))^{-1} A_i^{(j)}) \\ ((A^{(j)}(x))^{-1} A_i^{(j)})^T \bullet [(A^{(j)}(x))^{-1} A_k^{(j)}] &= Tr[((A^{(j)}(x))^{-1} A_i^{(j)})((A^{(j)}(x))^{-1} A_k^{(j)})] \end{aligned}$$

A new computation of the gradient and the Hessian of the weighted barrier function can be found in [5]. We have,

$$\begin{aligned} Tr((A^{(j)}(x))^{-1} A_i^{(j)}) &= Tr((A^{(j)}(x))^{-1/2} (A^{(j)}(x))^{-1/2} A_i^{(j)}) \\ &= Tr((A^{(j)}(x))^{-1/2} A_i^{(j)} (A^{(j)}(x))^{-1/2}) \end{aligned}$$

Now we have $((A^{(j)}(x))^{-1/2}A_i^{(j)}(A^{(j)}(x))^{-1/2})$ which is symmetric. Now the gradient of the weighted barrier function are computed below.

$$\nabla\phi_w(x)_i = \frac{wc_i}{(c^Tx_k^* - c^Tx + \epsilon)} - \sum_{j=1}^{q} Tr(A^{(j)}(x))^{-1/2}A_i^{(j)}(A^{(j)}(x))^{-1/2}) \qquad (6)$$

Now we look at the Hessian. We have,
$[(A^{(j)}(x))^{-1}A_i^{(j)}]^T \bullet [(A^{(j)}(x))^{-1}A_k^{(j)}] = \text{Tr } [((A^{(j)}(x))^{-1}A_i^{(j)})((A^{(j)}(x))^{-1}A_k^{(j)})]$
$=\text{Tr}[((A^{(j)}(x))^{-1/2}(A^{(j)}(x))^{-1/2}A_i^{(j)})((A^{(j)}(x))^{-1/2}(A^{(j)}(x))^{-1/2}A_k^{(j)})]$
$=\text{Tr}[((A^{(j)}(x))^{-1/2}A_i^{(j)}(A^{(j)}(x))^{-1/2})((A^{(j)}(x))^{-1/2}A_k^{(j)}(A^{(j)}(x))^{-1/2})]$
where $(A^{(j)}(x))^{-1/2}A_i^{(j)}(A^{(j)}(x))^{-1/2}$ and $((A^{(j)}(x))^{-1/2}A_k^{(j)}(A^{(j)}(x))^{-1/2}$ are both symmetric matrices. Now we have the new computation of the Hessian of the weighted barrier function below:

$$\nabla^2\phi_w(x)_{ik} = -\frac{wc_ic_k}{(c^Tx_k^* - c^Tx + \epsilon)^2} - \sum_{j=1}^{q} Tr[((A^{(j)}(x))^{-1/2}A_i^{(j)}(A^{(j)}(x))^{-1/2})((A^{(j)}(x))^{-1/2}A_k^{(j)}(A^{(j)}(x))^{-1/2})] \qquad (7)$$

In both the new and old ways of computing the gradient and Hessian, the inverses were calculated. Our experiments did not indicate an advantage of the new computations over the old computations. We will now consider another alternative. Recall that:

$$\begin{aligned} (A^{(j)}(x))^{-1}) \bullet A_i^{(j)} &= Tr((A^{(j)}(x))^{-1}A_i^{(j)}) \\ ((A^{(j)}(x))^{-1}A_i^{(j)})^T \bullet [(A^{(j)}(x))^{-1}A_k^{(j)}] &= Tr[((A^{(j)}(x))^{-1}A_i^{(j)})((A^{(j)}(x))^{-1}A_k^{(j)})] \end{aligned}$$

We will now use Guassian elimination to solve for $G^{(ij)}$.

$$A^{(j)}(x)G^{(ij)} = A_i \quad \forall i, j$$

Now we have,

$$\begin{aligned}
(A^{(j)}(x))^{-1}) \bullet A_i^{(j)} &= Tr((A^{(j)}(x))^{-1} A_i^{(j)}) \\
&= Tr(G^{(ij)}) \\
((A^{(j)}(x))^{-1} A_i^{(j)})^T \bullet [(A^{(j)}(x))^{-1} A_k^{(j)}] &= Tr[((A^{(j)}(x))^{-1} A_i^{(j)})((A^{(j)}(x))^{-1} A_k^{(j)})] \\
&= Tr(G^{(ij)} G^{(kj)}) \\
&= (G^{(ij)})^T \bullet G^{(kj)}
\end{aligned}$$

Now, we have avoided calculating the inverses in the expressions for the gradient and the Hessian.

3.4 Hessian Modification with Replacement

The Hessian matrix of the weighted barrier function ϕ_w can create numerical errors in the WAC-NEWTON algorithm when the Hessian is not positive definite. Recall that for a matrix to be positive definite all of the eigenvalues must be positive. The Hessian modification with replacement found in [6], appropriately exchanges the unsatisfactory eigenvalues for acceptable eigenvalues. This modification uses the spectral decomposition of the Hessian: $H = QDQ^T$ where the columns of Q are orthogonal eigenvectors of H and $D = \text{diag}(\lambda_1, \ldots, \lambda_n)$. The modified Hessian matrix is formed by eigenvalues given by:

$$\bar{\lambda}_i = \begin{cases} \lambda_i & \text{if } \lambda_\text{i} > 0 \\ \delta & \text{if } \lambda_\text{i} \leq 0 \end{cases} \tag{7}$$

Now in the WAC-NEWTON algorithm, replaces the original Hessian matrix for $\bar{H} = Q\bar{D}Q^T$ where $\bar{D} = \text{diag}(\bar{\lambda}_i, \ldots, \bar{\lambda}_n)$. Note that δ is chosen to be a small positive number. Observe that the matrix $\bar{H}$ is now positive definite.

3.5 Hessian Modification with Shifting

Similar to the above Hessian modification with replacement, this method modifies the Hessian matrix of the weighted barrier function so that it becomes positive semidefinite. Instead of replacing only the unsatisfactory eigenvalues, this method shifts all of the eigenvalues of the Hessian matrix H. The original Hessian of the weighted barrier function found in WAC-NEWTON is replaced by $\bar{H}$. The Hessian modification with shifting is found in [6] and is given by

$$\bar{H} = H + \tau I \text{ where, } \tau = \max(0, \delta - \lambda_{min}(H)).$$

where δ is a small positive number. Note that the eigenvalues of $\bar{H}$ are now all greater than or equal to δ.

4 Numerical Experiments

We have implemented the SDP-CUT in six ways. We used three different line search methods. We implemented the original SDP-CUT with the exact line search, the SDP-CUT with the bisection line search and the SDP-CUT with the Nesterov-Nemirovsky damping factor. We then modified the original SDP-CUT with exact line search in three different ways. We implemented the SDP-CUT with a new computation of the gradient and Hessian of the weighted barrier function. We im-

plemented the SDP-CUT with the Hessian modification of replacement of the eigenvalues and shifting of the eigenvalues. We will compare six varying implementations of the SDP-CUT. For each of the problems, the SDP-CUT parameters were $w = 7$ $\epsilon = 10^{-7}$, $STOL = 10^{-3}$, $WTOL = 10^{-6}$, $SMAX = 100$, and $WMAX = 100$.

For table 1, the SDP problems 1 through 10 were randomly generated with the number of of variables n from 2 to 20, the number of constraints q from 1 to 10, and the size of the matrices m from 1 to 20. The problems were generated in such a way that the origin is an interior point for all of them. Each variation of SDP-CUT was used to solve each problem. The SDP test problems truss1 and truss4 were chosen from the SDP library [8]. The respective number of variables n and number of constraints q were recorded. The number of SDP-CUT iterations and the run time (in seconds) were recorded for all of the problems. When the SDP-CUT implementation failed we used * in the respective column.

SDP	n	q	Iter	Exact Time(sec)	Bisection Time (sec)	NN Time (sec)	NewGradHess Time (sec)	Eig Replace Time (sec)	Eig Shift Time(sec)
1	17	19	16	9.81	9.84	79.87	5.13	8.64	8.65
2	18	19	16	10.42	10.86	83.10	5.71	9.78	9.77
3	18	8	15	4.61	5.06	36.19	2.64	4.51	4.50
4	16	20	14	8.41	9.12	58.32	4.54	8.00	7.94
5	12	10	11	2.17	2.41	16.01	1.30	2.08	2.10
6	6	2	7	0.17	0.23	0.96	0.15	0.18	0.20
7	18	19	16	10.09	10.57	86.97	5.67	10.01	9.87
8	20	5	12	3.05	3.25	26.81	1.41	3.01	2.96
9	6	20	8	1.01	1.53	5.28	0.68	1.02	1.03
10	19	9	13	5.01	6.87	43.02	2.38	5.09	5.05
truss1	6	7	11	*	3.09	5.58	*	*	1.00
truss4	12	7	16	*	5.57	*	*	3.92	*

Table 1: The original SDP-CUT compared to SDP-CUT with the five improvements. The sample semidefinite program problems in this table were randomly generated. They do not have a fixed m where m denotes the size of the matrices

We see in table 1 that for all of the randomly generated problems, the SDP-CUT with the bisection linesearch took more time than the SDP-CUT with exact linesearch. We can observe that with the SDP library problems truss1 and truss4, the SDP-CUT with exact linesearch failed. However, SDP-CUT with the bisection linesearch converged with truss1 and truss4.

In table 1, we now will look at the SDP-CUT with exact linesearch and the SDP-CUT with the Nesterov-Nemirovsky damping factor. The SDP problems 1 through 10 all converged in considerably less time when the SDP-CUT with exact line search was implemented than when the SDP-CUT with the Nesterov-Nemirovsky damping factor was implemented. We can observe that truss1 and truss4 converged when the SDP-CUT with the Nesterov-Nemirovsky damping factor was implemented. However, truss1 and truss4 failed to converge when the SDP-CUT with exact line search was implemented.

Looking at table1, when the SDP-CUT with the new computation of the gradient and Hessian was implemented, all of the SDP problems 1 to 10 took less time to run the algorithm than when SDP-CUT with exact linesearch was implemented. It is not surprising that truss1 and truss4 failed to converge when the SDP-CUT with the new computation of the gradient and Hessian was implemented. This algorithm still uses the exact line search in the WAC-NEWTON algorithm. Because truss1 and truss4 failed to converge with the original SDP-CUT with the exact linesearch, we would expect them to do the same with the new computation of the gradient and Hessian.

The original SDP-CUT took more time on almost all of the SDP problems 1 to 10 in table 1 when compared to the SDP-CUT where the Hessian is modified with the replacement of eigenvalues. Truss1 failed to converge when the SDP-CUT with

the replaced eigenvalues was implemented. Recall that this SDP-CUT modification still uses the exact line search so this is not surprising that this fails to converge. However, truss4 converges with SDP-CUT with replaced eigenvalues. The eigen values that had been replaced may have corrected the Hessian matrix enough to allow the exact line search to converge.

The SDP-CUT with the modified Hessian with shifted eigenvalues took less time to run in eight of the ten randomly generated problems in table 1 when compared to the original SDP-CUT. Truss1 converged with shifted eigenvalues but truss4 did not converge with the shifted eigenvalues.

We tested ten problems from the SDP library [8]. Truss1 and truss4 are among the ten problems. These were the only two problems that worked with SDP-CUT. Six of the problems were unbounded. It is not known if the other two are bounded or unbounded.

For table 2, the SDP problems were randomly generated. The origin is also an interior point for all of them. We varied the number of of variables n from 2 to 20, and the number of constraints q from 1 to 10. The size of the matrices m were fixed. The SDP problems 1 to 10 had a randomly generated fixed m from 1 to 20. The SDP problems 11 to 20 had randomly generated fixed m from 20 to 40. Each variation of SDP-CUT was used to solve each problem. It can be noted that each of the twenty SDP problems converged with all of the six methods.

All of the SDP problems in table 2, took more time to run the SDP-CUT algorithm with bisection linesearch than the time they took to run the original SDP-CUT with exact line search. In SDP problems 11 through 20 where the matrix sizes are larger, we can see that the SDP-CUT with bisection line search took significantly more time.

Compared to the original SDP-CUT with exact line search, the SDP-CUT with the Nesterov-Nemirovsky damping factor took more time to run 17 of the 20 SDP problems in table 2. Some of the SDP problems such as 1, 10, 17, and 18 all took significantly longer with the Nesterov-Nemirovsky damping factor.

SDP	m	n	q	Iter	Exact Time(sec)	Bisection Time (sec)	NN Time (sec)	NewGradHess Time (sec)	EigReplace Time (sec)	EigShift Time(sec)
1	14	17	10	11	4.42	4.93	35.36	5.45	4.42	4.42
2	9	3	3	7	0.14	2.07	0.45	0.20	0.13	0.12
3	17	7	7	7	0.68	1.04	3.90	0.88	0.67	0.67
4	9	7	8	9	0.71	1.04	4.32	1.09	0.70	0.71
5	4	11	4	13	0.82	0.92	6.84	1.32	0.88	0.81
6	19	6	2	6	0.23	0.37	1.04	0.27	0.21	0.20
7	16	3	5	4	0.14	0.31	0.31	0.18	0.13	0.12
8	16	5	10	7	0.56	0.99	2.30	0.83	0.54	0.54
9	11	18	1	10	0.92	1.88	8.43	0.55	0.94	0.93
10	8	8	8	7	0.59	1.87	3.74	1.01	0.60	0.57
11	40	15	2	9	2.33	4.23	1.37	1.08	2.43	2.25
12	36	14	3	9	2.70	4.71	1.54	1.17	2.55	2.61
13	23	12	5	11	2.04	4.40	0.83	1.08	2.03	2.01
14	25	4	8	7	0.57	2.47	2.07	0.37	0.55	0.56
15	24	16	1	7	0.75	2.03	4.98	0.44	0.71	0.70
16	24	14	9	8	3.25	5.12	21.84	1.63	3.19	3.22
17	37	15	5	9	5.84	8.41	34.79	2.06	5.31	5.30
18	33	19	4	8	4.29	6.15	33.69	1.85	4.24	4.25
19	39	10	1	5	0.46	1.77	2.80	0.26	0.47	0.43
20	29	16	5	9	3.51	5.34	26.81	1.62	3.55	3.49

Table 2: The original SDP-CUT compared to SDP-CUT with the five improvements. The sample semidefinite program problems in this table were randomly generated and have a fixed m.

The SDP-CUT with the new computation of the gradient and Hessian, took more time on almost all of the first ten problems where the size of the matrices vary from one to twenty. However, the problems eleven through twenty took significantly less time when implemented with the new computations over the old computations. We conclude that by not computing the inverses of the large matrices, we save time.

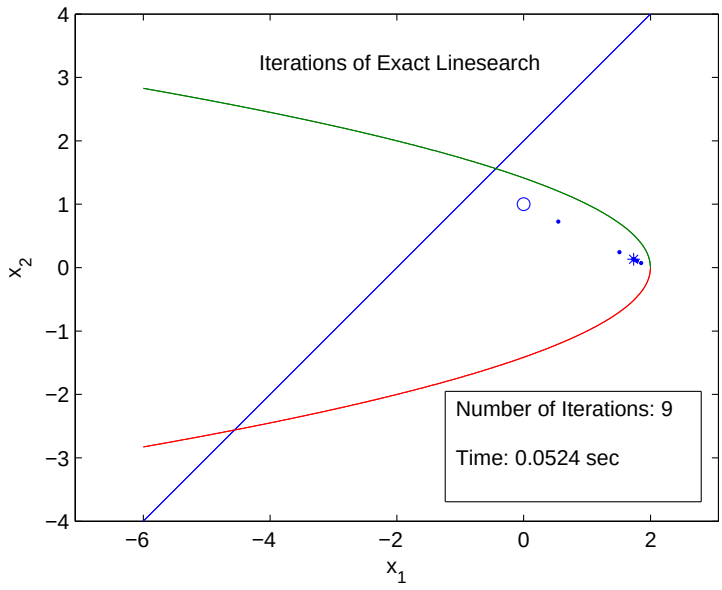
The SDP-CUT with the replaced eigenvalues took less time to run 15 of the 20 SDP problem than the original SDP-CUT. The SDP-CUT with the shifted eigenvalues took less time to run 16 of the 20 SDP problems than the original SDP-CUT.

Figures 2 to 5 show iterations of the algorithms. The initial starting point is shown as o. The final iteration is shown as a star and the iterations in between are shown as a dot.

Figure 2 displays the feasible region of the example. The graphic shows the iterations of the exact line search used to find the weighted analytic center of the example in the WAC-NEWTON algorithm.

Figure 3 also displays the feasible region of the example. This graphic shows

Figure 2:



the iterations of the bisection line search that is implemented to find the weighted analytic center of the example in the WAC-NEWTON algorithm. The exact line search only took nine iterations to converge. However, the bisection linesearch took

21 iterations to converge. The bisection linesearch also took about twice as much time to run as the exact linesearch took.

Figure 3:

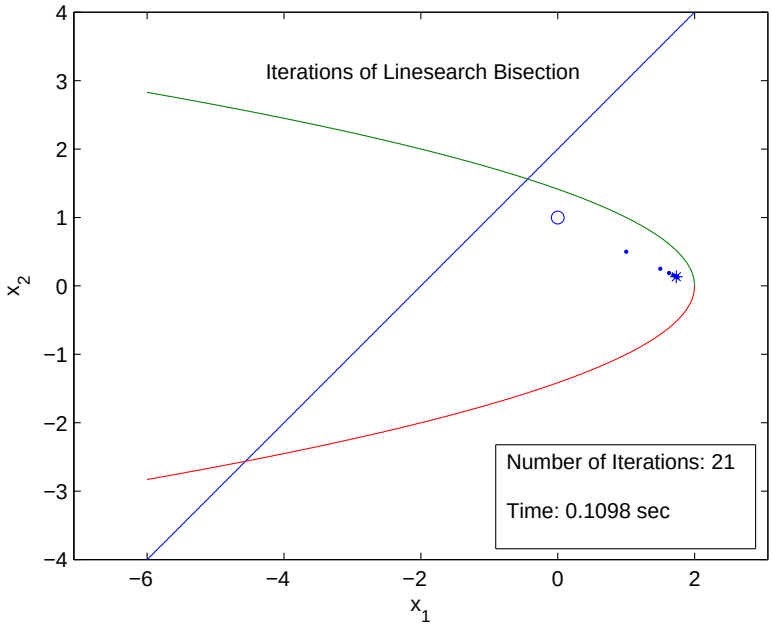


Figure 4 displays the feasible region of the example. It shows the iterations of the WAC-NEWTON algorithm with the exact implemented in the first iteration of the SDP-CUT.

Figure 5 shows the iterations of the WAC-NEWTON algorithm with the Nesterov-Nemirovsky damping facto in the first iteration of the SDP-CUT. There were 81 iterations using the Nesterov-Nemirovsky damping factor compared to 13 iterations with exact line search. The exact line search had a much faster running time.

Figure 4: Below is the feasible region of the example. This graphic shows the iterations of the WAC-NEWTON with the exact line search.

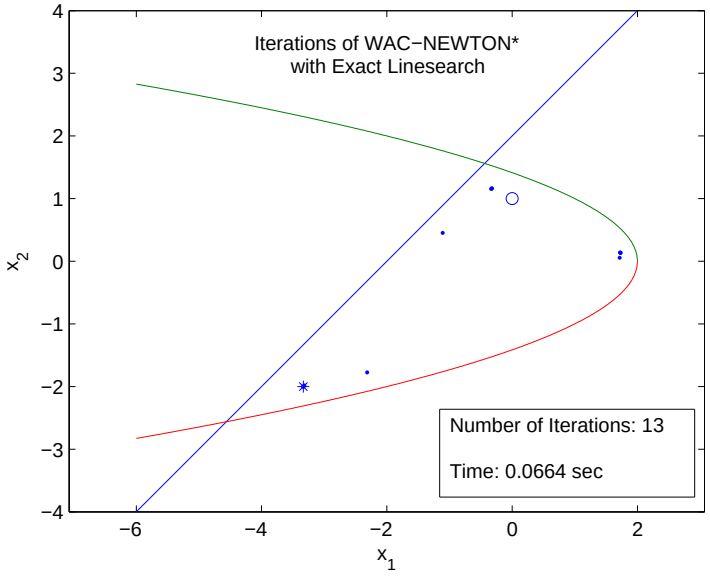
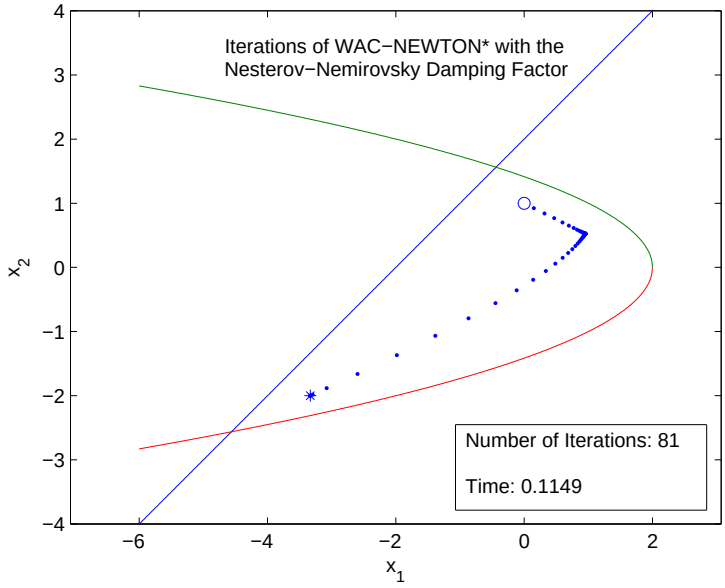


Figure 5: Below is the feasible region of the example. This graphic shows the iterations of the WAC-NEWTON with the Nesterov-Nemirovsky Damping Factor.



5 Conclusion

This paper presented modifications of the interior point algorithm SDP-CUT for solving SDP problems. The goal was to improved the accuracy and time in solving the SDP problems. We looked into the linesearch methods of bisection and Nesterov-Nemirovsky's damping factor and compared them to the exact line search that was used int he original SDP-CUT. We conclude that the exact linesearch is a better method to use for the SDP-CUT. The SDP-CUT with exact linesearch generally took less time. We examined how modifying the Hessian would impact the SDP-CUT. One way of modifying the Hessian replaces the negative eigenvalues found in the spectoral decomposition. Another modification of the Hessian shifts all of its eigenvalues. Both of the modifications to the Hessian improved the SDP-CUT algorithm. The SDP-CUT algorithm took less time with the replaced eigenvalues and the shifted eigenvalues than the original SDP-CUT. The SDP-CUT with the shifted eigenvalues seemed to improve running time of the SDP-CUT in more test problems than the replaced eigenvalues. We also examined the effect of a more efficient way of computing the gradient and Hessian of the barrier function. The new computation of the gradient and the Hessian saved a good amount of time for the problems with larger matrices. We conclude that the SDP-CUT with the exact linesearch is more efficient than the other linesearch methods examined. The SDP-CUT with the shifted eigenvalues should be implemented because it improved the running time of most of the test problems. Also, the SDP-CUT with the new computation of the gradient and Hessian should be implemented.

The SDP-CUT has more room for improvement. Currently to implement the SDP-CUT, we must know an initial starting point that is inside the feasible region.

However, it may not always be reasonable to know a point inside the feasible region. For this reason, it is important to investigate the infeasible Newton method to compute the analytic center from an infeasible point. In the case of linear constraints, this was described in [4]. We intend to implement this and to extend the technique to handle general LMI constraints.

Another issue is how to deal with problems with unbounded feasible regions. It might be useful to use a box that is known to intersect the feasible region and contains the optimal solution.

6 Acknowledgment

The author would like to thank the National Science Foundation (NSF) for the opportunity to start initial work on this paper through their funding of Research Experiences for Undergraduates (REU) program at Northern Arizona University.

References

[1] Etienne de Klerk, "Aspects of Semidefinite Programmingl", *Algorithmica*, vol. 65, 2002, pp. 1-3.

[2] L. Vandenberghe and S. Boyd, "Semidefinite Programming", *SIAM Review,* March 1996.

[3] J. Machacek and S. Jibrin, "An Interior Point Method for Solving Semidefinite Programs Using Cutting Planes and Weighted Analytic Centers", Northern Arizona University REU Report, 2010.

[4] L. Vandenberghe and S. Boyd, "Convex Optimization", Cambridge University Press, New York 2004.

[5] S. Boyd and L. Ghaoui, "Method of Centers For Minimizing Generalized Eigenvalues", Standford University, Standford, California 1993.

[6] J. Nocedal and S. Wright, "Numerical Optimization", Springer Science Business Media, 2006, pp. 49-51.

[7] M. Grant and S. Boyd "CVX: Matlab software for disciplined programming", version 1.21, 2011.

[8] B. Borchers and L. Vandenberghe, "SDPLIB 1.2, a library of semidefinite programming test problems", *Optimization Methods and Software.* Vol. 11 & 12, 1999, pp. 683-690. Available at *http://www.nmt.edu/borchers/sdplib.html.*