

Research Article

An Interior Point Method for Solving Semidefinite Programs Using Cutting Planes and Weighted Analytic Centers

John Machacek¹ and Shafiu Jibrin²

¹ Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA

² Department of Mathematics and Statistics, Northern Arizona University, Flagstaff, Arizona 86011-5717, USA

Correspondence should be addressed to Shafiu Jibrin, shafiu.jibrin@nau.edu

Received 11 October 2011; Revised 10 May 2012; Accepted 24 May 2012

Academic Editor: James Buchanan

Copyright © 2012 J. Machacek and S. Jibrin. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We investigate solving semidefinite programs (SDPs) with an interior point method called SDP-CUT, which utilizes weighted analytic centers and cutting plane constraints. SDP-CUT iteratively refines the feasible region to achieve the optimal solution. The algorithm uses Newton's method to compute the weighted analytic center. We investigate different stepsize determining techniques. We found that using Newton's method with exact line search is generally the best implementation of the algorithm. We have also compared our algorithm to the SDPT3 method and found that SDP-CUT initially gets into the neighborhood of the optimal solution in less iterations on all our test problems. SDP-CUT also took less iterations to reach optimality on many of the problems. However, SDPT3 required less iterations on most of the test problems and less time on all the problems. Some theoretical properties of the convergence of SDP-CUT are also discussed.

1. Introduction

We consider the semidefinite programming problem (SDP) as in [1]:

$$\text{minimize } c^T x, \quad (1.1)$$

$$\text{subject to } A^{(j)}(x) \geq 0 \quad \text{for } j = 1, \dots, q, \quad (1.2)$$

where $A^{(j)}(x) := A_0^{(j)} + \sum_{i=1}^n x_i A_i^{(j)}$. Also, $x \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and each $A_i^{(j)}$ is an $m_j \times m_j$ symmetric matrix. The constraint $A^{(j)}(x) \geq 0$ is known as a *linear matrix inequality* or LMI.

For a given matrix A , $A \geq 0$ denotes that A is positive semidefinite. The notation > 0 is used when a matrix is positive definite. Semidefinite Programs (SDPs) are *convex* optimization problems [2], and they generalize many other convex optimization problems including linear programming. SDPs have applications in engineering and combinatorial optimization and other fields (see [2, 3]).

The dual of problem (1.1) is the optimization problem [1]:

$$\text{maximize} \quad - \sum_{j=1}^q A_0^{(j)} \bullet Z_j \quad (1.3)$$

$$\text{subject to} \quad \sum_{j=1}^q A_i^{(j)} \bullet Z_j = c_i \quad \text{for } i = 1, \dots, n \quad (1.4)$$

$$Z_j \geq 0 \quad \text{for } j = 1, \dots, q. \quad (1.5)$$

In the above, $A \bullet B$ denotes the matrix dot product. If $A = [a_{ij}]$ and $B = [b_{ij}]$ are matrices of size m , then $A \bullet B = \sum_{i=1}^m \sum_{j=1}^m a_{ij} b_{ij} = \text{Tr}[AB^T]$, where Tr denotes the *trace*. We let $\mathcal{R} = \{x \mid A^{(j)}(x) \geq 0 \text{ for all } j = 1, \dots, q\}$ the *feasible region* and $\text{int}(\mathcal{R}) = \{x \mid A^{(j)}(x) > 0 \text{ for all } j = 1, \dots, q\}$ the interior of the feasible region. The set $\text{int}(\mathcal{R})$ is precisely the set of all *strictly feasible points*. Let p^* be the optimal solution to the primal problem (1.1), and let d^* be the optimal solution to the dual problem (1.3). The duality gap $p^* - d^*$ is zero if either SDP (1.1)-(1.2) or its dual (1.3)-(1.5) is strictly feasible [2].

Several interior point methods have been developed for solving SDPs (see [2, 4, 5]). When developing our interior point method in this paper, we assume a strictly feasible interior point x_0 is known. There exist methods for finding feasible or near feasible points (see [6–8]). The algorithm we develop uses *cutting planes* and *weighted analytic centers*, and it iteratively refines the feasible region until the weighted analytic center approaches the optimal solution. We call our algorithm SDP-CUT. The cutting plane technique was pioneered by Gomory [9], Kelley [10], and also Cheney and Goldstein [11] for solving integer programming problems. An algorithm similar to our SDP-CUT for linear programs (LPs) was proposed by Renegar in his 1988 paper [12].

SDP-CUT is implemented using Newton's method and different line search techniques. We found that Newton's method with exact line search is the best implementation of our algorithm. The effects of a weight vector $w \in \mathbb{R}_+$ on the algorithm are also studied. A larger weight yields a faster and more accurate solution in theory, but in practice, too large a weight may cause numerical errors. We also experienced numerical errors in computing the Hessian matrices in SDP-CUT, when solving very large problems.

Since finding an interior point for an SDP problem is equivalent to solving another SDP problem, we decided to consider test problems with a known interior point and used that point as a starting point for SDP-CUT. We find SDPT3 to be an ideal method for comparison with SDP-CUT because it is known to be efficient and it allows the user to input a starting point. We found SDP-CUT was closer to the actual solution than SDPT3 for the initial iterations on all our test problems. SDP-CUT seems to slow down during later iterations to reach optimality. On the other hand, SDPT3 took less time on all the problems and less iterations on most of them. All codes were written in MATLAB version 7.9.0.

2. Weighted Analytic Center

This section discusses weighted analytic center as given in [13] and how to compute it. Other notions of weighted center for semidefinite programming are described in [14].

Given a weight vector $\omega \in \mathbb{R}_+^n$, the *weighted barrier function* for our system of LMIs (1.2) is defined as follows:

$$\phi_\omega(x) = \begin{cases} -\sum_{j=1}^q \omega_j \log \det[A^{(j)}(x)] & \text{if } A^{(j)}(x) \succ 0 \quad \forall j = 1, \dots, q, \\ +\infty & \text{otherwise.} \end{cases} \quad (2.1)$$

Note that as x approaches the boundary of the feasible region, $\phi_\omega(x)$ approaches ∞ . We assume the set $\{\text{diag}(A_1^{(1)}, \dots, A_1^{(q)}), \text{diag}(A_2^{(2)}, \dots, A_2^{(q)}), \text{diag}(A_n^{(1)}, \dots, A_n^{(q)})\}$ is linearly independent. The function ϕ_ω is *analytic* and *strictly convex* over \mathcal{R} (see [13, 15]). The *weighted analytic center* is defined to be $x_{ac}(\omega) = \text{argmin} \{\phi_\omega(x) \mid x \in \mathbb{R}^n\}$. We call $x_{ac}(\mathbf{1})$ the *analytic center*, where $\mathbf{1} = [1, 1, \dots, 1]$. In the case of linear constraints and some other more general LMIs, each weight pushes the analytic center away from the boundary of the corresponding constraint. The gradient $\nabla \phi_\omega(x)$ and the Hessian $\nabla^2 \phi_\omega(x)$ are given by the following: for $i = 1, \dots, n$ and $k = 1, \dots, n$

$$\begin{aligned} (\nabla \phi_\omega(x))_i &= -\sum_{j=1}^q \omega_j \left(A^{(j)}(x) \right)^{-1} \bullet A_i^{(j)}, \\ (\nabla^2 \phi_\omega(x))_{ik} &= \sum_{j=1}^q \omega_j \left[\left(A^{(j)}(x) \right)^{-1} A_i^{(j)} \right]^T \bullet \left[\left(A^{(j)}(x) \right)^{-1} A_k^{(j)} \right]. \end{aligned} \quad (2.2)$$

We will use the following specific example, SDP (2.3), to describe our method:

$$\text{SDP: minimize } z = x_1 + 2x_2$$

subject to

$$A^{(1)}(x) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} + x_1 \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2-x_1 & x_2 \\ x_2 & 1 \end{bmatrix} \succeq 0, \quad (1) \quad (2.3)$$

$$A^{(2)}(x) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} + x_1 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + x_2 \begin{bmatrix} -1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} x_1-x_2 & 0 \\ 0 & x_1 \end{bmatrix} \succeq 0. \quad (2)$$

Weighted analytic centers for the LMI system (1.2) can be computed with the WAC-NEWTON algorithm described below. (Algorithm 1).

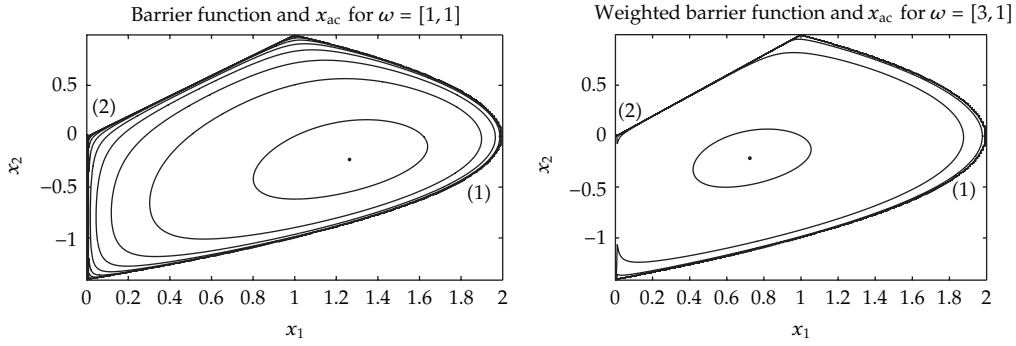
An efficient way to calculate the search vector s_l in WAC-NEWTON is by finding the Cholesky factorization of $\nabla^2 \phi_\omega(y_l)$ and using it to solve the linear system $\nabla^2 \phi_\omega(y_l) s_l = -\nabla \phi_\omega(y_l)$. This is how s_l is calculated in the SDP-CUT algorithm to be discussed in the next section and throughout this paper. Figure 1 shows the contours of $\phi_\omega(x)$, the effect of the weight vector $\omega = [3, 1]$ on the barrier function, and the weighted analytic center. The figure uses the feasible region defined by SDP (2.3) and the associated barrier function. The weight of 3 on constraint (1) pushes the analytic center away from the boundary of constraint (1).

```

INPUT: point  $y_0 \in \text{int}(\mathcal{R})$ , weight vector  $\omega \in \mathbb{R}^d$ , tolerance WTOL > 0, and maximum
number of iterations MAX
Set  $l = 0$ 
while  $l < \text{MAX}$  do
  1. Compute the direction vector  $s_l = -(\nabla^2 \phi_\omega(y_l))^{-1} (\nabla \phi_\omega(y_l))$ 
  2. Compute the Newton decrement  $d = \sqrt{s_l^T (\nabla^2 \phi_\omega(y_l)) s_l}$ 
  3. Compute stepsize  $\alpha_l$ 
  4.  $y_{l+1} = y_l + \alpha_l s_l$ 
  if  $d < \text{WTOL}$  then
    break while
  5.  $l \leftarrow l + 1$ 
OUTPUT:  $x_{\text{ac}}(\omega) = y_l$ 

```

Algorithm 1: WAC-NEWTON.

Figure 1: $\phi_\omega(x)$ contours and $x_{\text{ac}}(\omega)$ with various weights ω .

3. The SDP-CUT Algorithm

This section describes the development of SDP-CUT. We also discuss WAC-NEWTON*, which implements Newton's method for finding the *weighted analytic center* for the new system defined in SDP-CUT. The section finishes with algorithms for computing the *Newton stepsize* (CONSTANT, ELS, and BACKTRACKING).

Refer again to the example SDP (2.3). Here, we illustrate one iteration of SDP-CUT. Figure 2 shows the setup. We have the feasible region, an initial point $x_0^* = [1.2633, -0.2292]^T$, and a cutting constraint of the form $c^T x_k^* - c^T x + \epsilon \geq 0$ that accompanies the point. Figure 3 shows the movement of the point x_0^* to $x_1^* = x_{\text{ac}}(1)$, the (weighted) analytic center of our new feasible region made up of the cutting constraint and the original LMI constraints. Figure 3 also has the contour lines of the barrier function. The weight on the cutting constraint can be changed from 1 to other larger values.

Given a system of LMI's (1.2) and an objective function as in the primal SDP problem (1.1), we can numerically solve the problem by iteratively reducing the feasible set. Denote the current feasible region determined by our system of LMIs by \mathcal{R}_k , and suppose we know a strictly feasible point x_k^* in \mathcal{R}_k . Initially, we set $\mathcal{R}_0 = \mathcal{R}$. We can find a new feasible region $\mathcal{R}_{k+1} \subset \mathcal{R}_k$, where for all $x \in \mathcal{R}_{k+1}$, $c^T x \leq c^T x_k^*$. We do this by adding a new constraint cut $c^T x_k^* - c^T x + \epsilon \geq 0$. The ϵ is added to $c^T x_k^*$ to ensure that x_k^* is "strictly feasible" in the new

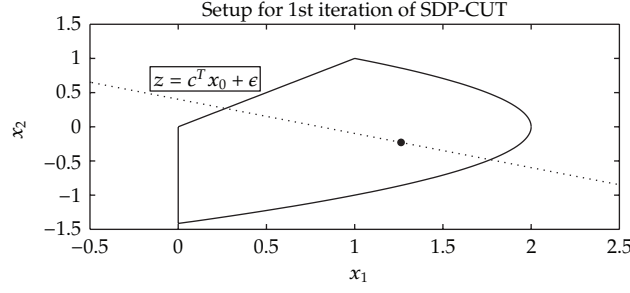


Figure 2: The first cutting constraint and an initial point x_0^* .

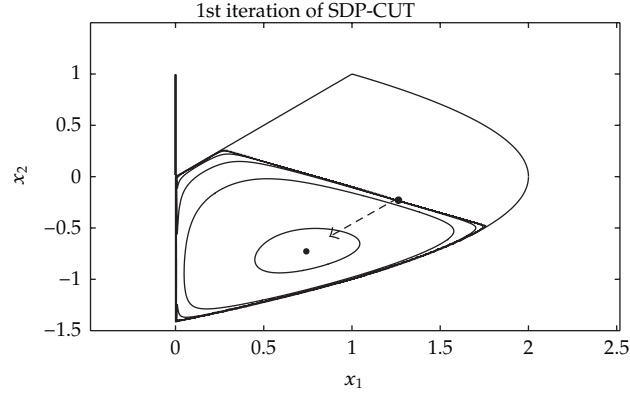


Figure 3: The weighted analytic center of new feasible region.

system. Given weight $w \in \mathbb{R}_+$, we define a new barrier function to account for this new constraint:

$$\phi_w^*(x) = -w \log [c^T x_k^* - c^T x + \epsilon] - \sum_{j=1}^q \log \det [A^{(j)}(x)]. \quad (3.1)$$

The gradient $\nabla \phi_w^*(x)$ and the Hessian $\nabla^2 \phi_w^*(x)$ for this new barrier function are given by the following: for $i = 1, \dots, n$ and $k = 1, \dots, n$:

$$(\nabla \phi_w^*(x))_i = \frac{w c_i}{(c^T x_k^* - c^T x + \epsilon)} - \sum_{j=1}^q \left(A^{(j)}(x) \right)^{-1} \bullet A_i^{(j)}, \quad (3.2)$$

$$\left(\nabla^2 \phi_w^*(x) \right)_{ik} = \frac{w c_i c_k}{(c^T x_k^* - c^T x + \epsilon)^2} + \sum_{j=1}^q \left[\left(A^{(j)}(x) \right)^{-1} A_i^{(j)} \right]^T \bullet \left[\left(A^{(j)}(x) \right)^{-1} A_k^{(j)} \right]. \quad (3.3)$$

Remark 3.1. Note that in the definition of $\phi_w^*(x)$ in (3.1), weight $[1, 1, \dots, 1]$ is used on the original LMI constraints. The weight w is used only on the new cutting constraint in order to push our point toward the optimal solution. (see Algorithm 2).

```

INPUT:  $x_0^* \in \text{int}(\mathcal{R})$ , weight  $w \in \mathbb{R}_+$ ,  $\epsilon > 0$ , STOL  $> 0$ , and MAX
Set  $k = 0$ 
While  $k < \text{MAX}$  do
  1. Compute cutting plain constraint  $c^T x - c^T x_k^* + \epsilon \geq 0$ 
  2. Let  $x_{k+1}^*$  be the weighted analytic center of new system with barrier function  $\phi_w^*(x)$ 
    (3.1) 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  $\partial_k < \text{STOL}$  then
    break while
  3.  $k \leftarrow k + 1$ 
OUTPUT:  $x_{\text{cut}}^* = x_k^*$ ,  $p_{\text{cut}}^* = c^T x_k^*$ 

```

Algorithm 2: SDP-CUT.

If successful, SDP-CUT terminates with an optimal solution x_{cut}^* and optimal objective function value p_{cut}^* . Rather than moving the plane in Step 1 of SDP-CUT, the point x_k^* could also be moved in the direction of $-c$ to obtain a different starting point $x_k^* - \gamma c$ for some small $\gamma > 0$, instead of x_k^* . However, this can be problematic as we approach the optimal solution and the feasible region gets small. The point $x_k^* - \gamma c$ may pass over the entire remaining feasible region and thus fail to be feasible. If instead, we move the plane as originally suggested in SDP-CUT, x_k^* will always be strictly feasible. Note that since the objective is to maximize $c^T x$ and x_k^* is an interior point, then $-c$ is a feasible direction from x_k^* .

We denote by WAC-NEWTON*, the WAC-NEWTON algorithm applied to $\phi_w^*(x)$ (3.1) for determining x_{k+1}^* in the SDP-CUT algorithm. WAC-NEWTON* will return the weighted analytic center of the current feasible region, which will be our next iterate for SDP-CUT. In WAC-NEWTON*, the stepsize α_l can be computed in a variety of different ways as discussed in the next section.

3.1. Line Searches: Computing the Newton Stepsize in WAC-NEWTON* Algorithm

We describe different options for computing the Newton stepsizes in the WAC-NEWTON* algorithm. The algorithm first computes the direction vector s_l . The Newton stepsize α_l determines how far we should move in the direction of s_l from the point y_l .

The pure Newton's method uses a constant stepsize $\alpha_l = 1$. We will refer to this method of choosing the stepsize as "CONSTANT." The CONSTANT algorithm has the advantage of not using computational time in deciding what stepsize to use. The disadvantage of Newton's method with CONSTANT is that it usually results in the need to perform more iterations of WAC-NEWTON*, and it is possible to move out the feasible region. To get α_l with exact line search, we solve the one-dimensional optimization problem:

$$\text{minimize}\{g(\alpha) \mid \alpha > 0\}, \quad (3.4)$$

where $g(\alpha) = \phi_w^*(y_l + \alpha s_l)$. Solving (3.4) is relatively easy and may cut the number of WAC-NEWTON* iterations in SDP-CUT, which are computationally harder in comparison

$$\begin{aligned} g(\alpha) &= \phi_w^*(y_l + \alpha s_l) \\ &= -w \log [c^T x_k^* - c^T (y_l + \alpha s_l) + \epsilon] - \sum_{j=1}^q \log \det [A^{(j)}(y_l + \alpha s_l)]. \end{aligned} \quad (3.5)$$

Let $a_l = w \log [c^T x_k^* - c^T (y_l + \alpha s_l) + \epsilon]$. Then

$$\begin{aligned} g(\alpha) &= -a_l - \sum_{j=1}^q \log \det [A^{(j)}(y_l + \alpha s_l)] \\ &= -a_l - \sum_{j=1}^q \log \det \left[A^{(j)}(y_l) + \sum_{i=1}^n (\alpha s_l)_i A_i^{(j)} \right]. \end{aligned} \quad (3.6)$$

Let $M_0^{(j)} = A^{(j)}(y_l)$ and $M_s^{(j)} = \sum_{i=1}^n (s_l)_i A_i^{(j)}$. Then

$$g(\alpha) = -a_l - \sum_{j=1}^q \log \det [M_0^{(j)} + \alpha M_s^{(j)}]. \quad (3.7)$$

Also,

$$\begin{aligned} g(\alpha) &= -a_l - \sum_{j=1}^q \log \det [M_0^{(j)} + \alpha M_s^{(j)}] \\ &= -a_l - \sum_{j=1}^q \log \det \left[\left(M_0^{(j)} \right)^{1/2} \left(I + \alpha \left(M_0^{(j)} \right)^{-1/2} M_s^{(j)} \left(M_0^{(j)} \right)^{-1/2} \right) \left(M_0^{(j)} \right)^{1/2} \right] \\ &= -a_l - \sum_{j=1}^q \log \left[\det \left[\left(M_0^{(j)} \right)^{1/2} \right] \det \left[\left(I + \alpha \left(M_0^{(j)} \right)^{-1/2} M_s^{(j)} \left(M_0^{(j)} \right)^{-1/2} \right) \right] \right. \\ &\quad \left. \times \det \left[\left(M_0^{(j)} \right)^{1/2} \right] \right] \\ &= -a_l - \sum_{j=1}^q \log \left[\det \left[\left(M_0^{(j)} \right)^{1/2} \right]^2 \det \left[\left(I + \alpha \left(M_0^{(j)} \right)^{-1/2} M_s^{(j)} \left(M_0^{(j)} \right)^{-1/2} \right) \right] \right] \\ &= -a_l - \sum_{j=1}^q \log \left[\det \left[\left(M_0^{(j)} \right) \right] \det \left[\left(I + \alpha \left(M_0^{(j)} \right)^{-1/2} M_s^{(j)} \left(M_0^{(j)} \right)^{-1/2} \right) \right] \right] \\ &= -a_l - \sum_{j=1}^q \log \det \left[\left(M_0^{(j)} \right) \right] - \sum_{j=1}^q \log \det \left[\left(I + \alpha \left(M_0^{(j)} \right)^{-1/2} M_s^{(j)} \left(M_0^{(j)} \right)^{-1/2} \right) \right]. \end{aligned} \quad (3.8)$$

Let $\lambda_i^{(j)}$ be an eigenvalue of $(M_0^{(j)})^{-1/2} M_s^{(j)} (M_0^{(j)})^{-1/2}$. Then

$$g(\alpha) = -a_l - \sum_{j=1}^q \log \det \left[\left(M_0^{(j)} \right) \right] - \sum_{j=1}^q \sum_{i=1}^{m_j} \log \left[1 + \alpha \lambda_i^{(j)} \right]. \quad (3.9)$$

The function $g(\alpha)$ is convex function over \mathbb{R} , and the exact stepsize α_l can be found using the Newton's method. Equations (3.7) and (3.9) give two different ways of computing $g(\alpha)$, when finding the exact stepsize. If one decides to use (3.7), the first and second derivatives of $g(\alpha)$ are required and given by

$$g'(\alpha) = \frac{w c^T s_l}{c^T x_k^* - c^T (y_l + \alpha s_l) + \epsilon} - \sum_{j=1}^q \left(M_0^{(j)} + \alpha M_s^{(j)} \right)^{-1} \bullet M_s^{(j)}, \quad (3.10)$$

$$g''(\alpha) = - \frac{w (c^T s_l)^2}{(c^T x_k^* - c^T (y_l + \alpha s_l) + \epsilon)^2} + \sum_{j=1}^q \left[\left(M_0^{(j)} + \alpha M_s^{(j)} \right)^{-1} M_s^{(j)} \right]^T \bullet \left[\left(M_0^{(j)} + \alpha M_s^{(j)} \right)^{-1} M_s^{(j)} \right]. \quad (3.11)$$

If (3.9) is used, the derivatives are simply given by

$$g'(\alpha) = \frac{w c^T s_l}{c^T x_k^* - c^T (y_l + \alpha s_l) + \epsilon} - \sum_{j=1}^q \sum_{i=1}^{m_j} \frac{\lambda_i^{(j)}}{1 + \alpha \lambda_i^{(j)}}, \quad (3.12)$$

$$g''(\alpha) = \frac{w (c^T s_l)^2}{(c^T x_k^* - c^T (y_l + \alpha s_l) + \epsilon)^2} - \sum_{j=1}^q \sum_{i=1}^{m_j} \frac{(\lambda_i^{(j)})^2}{(1 + \alpha \lambda_i^{(j)})^2}. \quad (3.13)$$

We will use ELS-MAT to denote the exact line search computations done using (3.7), and ELS-EIG if computations are done from (3.9).

In addition to constant stepsize = 1 and exact line search, another way that was considered in computing the Newton stepsize was *backtracking*. This method involves starting with a stepsize ≥ 1 and then, decreasing the stepsize until a stopping condition is met (see [5, 16]). This technique guarantees a sufficient decrease in $g(\alpha)$, often starting from $\alpha = 1$, in practice. (see Algorithm 3).

It is known that Newton's method converges quadratically close to the solution. In our case, WAC-NEWTON* converges rapidly for starting points that are not close to the boundary of the feasible region \mathcal{R}_k . Sometimes, we encounter difficulty when the starting point is too close to the boundary. Note that each time we make a new cut, our starting point is near the boundary. It is also the case, when SDP-CUT iterates approaches optimality. When close to the boundary, our direction vector s often is very small, and thus a stepsize of $\alpha = 1$ may not make much progress. So, in these cases, the CONSTANT stepsize may spend many iterations, while making little progress, and each iteration wasting gradient and Hessian computations. Using the ELS algorithm, we find the proper stepsize and move out


```

INPUT:  $0 < \beta < 1$  and  $0 < \gamma < 0.5$ 
Set  $\alpha = 1$ 
while  $g(\alpha) > g(0) + \alpha\gamma(\nabla\varphi_w(y_l))^T s_l$  do
   $\alpha \leftarrow \beta\alpha$ 
OUTPUT:  $\alpha$ 

```

Algorithm 3: Backtracking.

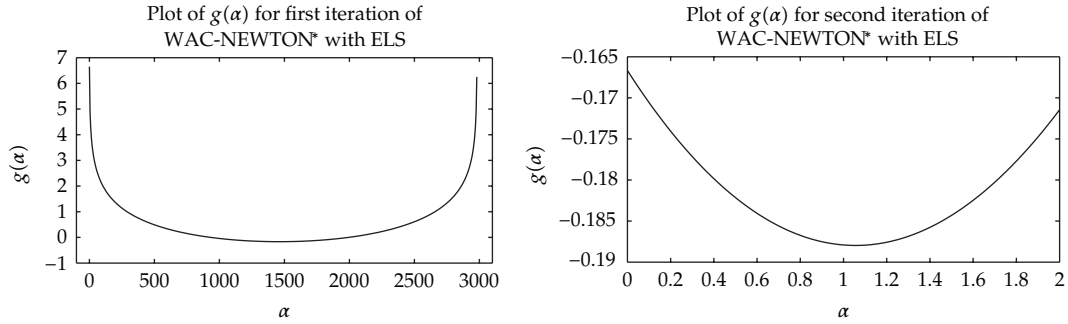


Figure 4: Need for different stepsizes.

of this problem area in just one iteration. Consider the plots in Figure 4, which show the situation described above. Here, again, we are using SDP Example 2.3 and the point shown in Figure 2.

As we can see from Figure 4 on the left, the best choice of stepsize is much greater than 1. In Figure 4 on the right, we can see that we have now moved into an area where a stepsize of 1 is reasonable. BACKTRACKING will not help with this problem. BACKTRACKING only helps when the optimal stepsize is less than 1. If we try to adapt BACKTRACKING to help in the circumstance described above, we must initialize the BACKTRACKING stepsize to a large number, which creates two problems. First, when the optimal stepsize is close to 1, or smaller, we will waste time backtracking. Secondly, if we use a large initial stepsize, this may cause BACKTRACKING to send our point outside the feasible region, causing our algorithm to diverge. As it will be further shown in the next section, it does appear that ELS has a positive effect on SDP-CUT, while BACKTRACKING does not. Figure 5 has plots comparing how far CONSTANT and ELS move our point towards the optimal solution at each iteration. The plots highlight the “wasted iteration” problem with CONSTANT, which occurs when the optimal stepsize is much greater than 1.

4. Experiment I: SDP-CUT Implementations

We have implemented SDP-CUT in four ways, varying in how the Newton stepsize is computed: CONSTANT, ELS-MAT, ELS-EIG, and BACKTRACKING. We will compare the performance of these four implementations against various test problems. For each problem, the SDP-CUT parameters were set at $\epsilon = 10^{-6}$, $\text{STOL} = 10^{-12}$, $\text{WTOL} = 10^{-10}$, $w = 7$ and $\text{MAX} = 100$.

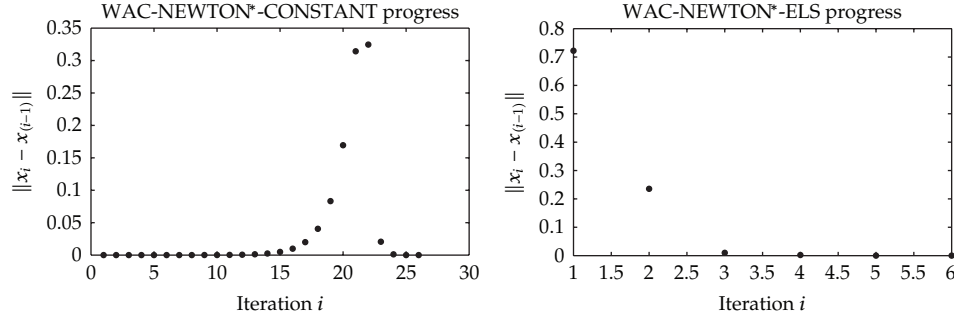


Figure 5: The “wasted iteration” problem.

4.1. Performance as q Varies

Here, we left the number of variables constant at $n = 2$ and varied the number of constraints q from 2 to 10. The matrices have their sizes fixed at $m_j = 5$ for all j . For each value of q , 10 SDP problems were randomly generated, and SDP-CUT was used to solve each problem. The number of SDP-CUT iterations (which we also call WAC-NEWTON* iterations) and run time for SDP-CUT were recorded for each problem. Figures 6 and 7 give plots of the averages with q on the horizontal axis. The vertical axis in Figure 6 contains the total SDP-CUT iterations. In Figure 7, the vertical axis gives the total run time of SDP-CUT in seconds.

Figure 6 shows there is not a correlation between the number of LMI constraints and the number of SDP-CUT iterations (WAC-NEWTON* iterations) needed to find the optimal solution. Figure 6 also shows that ELS-EIG and ELS-MAT both effectively reduce the number of SDP-CUT iterations needed, while BACKTRACKING has about the same number of iterations as CONSTANT. Figure 7 shows that in the case of $n = 2$, ELS-MAT and ELS-EIG run the fastest, while BACKTRACKING runs far slower than anything else. Time increases as q increases due to the gradient and Hessian computations. As we can see in formulas (3.2) and (3.3), for each constraint, we must perform matrix inverses, dot products, and multiplications. This makes each iteration computationally harder, but does not affect the iterations needed as is seen in Figure 6. BACKTRACKING’s time increases the most with q due to the eigenvalue computation needed for the stopping condition. BACKTRACKING performs these extra computations, but SDP-CUT does not benefit from them, and consequently BACKTRACKING has the highest run time.

4.2. Performance as n Varies

Here, we left the number of constraints constant at $q = 3$ and varied the number of variables n from 2 to 10. The matrices sizes were set at $m_j = 5$ for all j . For each value of n , 10 SDP problems were randomly generated, and SDP-CUT was used to solve each problem. The number of SDP-CUT iterations (WAC-NEWTON* iterations) and run time for SDP-CUT were recorded for each problem. Figures 8, 9, and 10 are plots of the averages with n on the horizontal axis. The vertical axis contains the SDP-CUT iterations (WAC-NEWTON* iterations) needed in Figure 8. The vertical gives the run time of SDP-CUT in seconds in Figure 9. There is also a third plot, Figure 10, which shows n on the horizontal axis and the average time required to perform an iteration of SDP-CUT on the vertical axis.

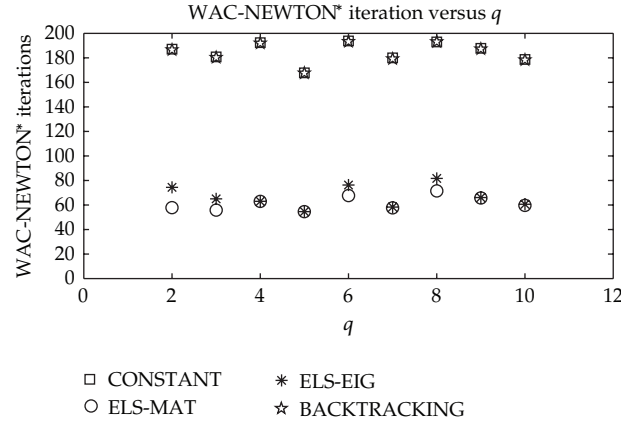


Figure 6: SDP-CUT iterations against the number of constraints q .

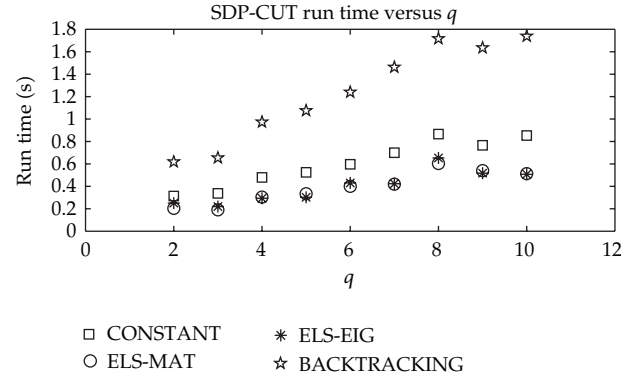


Figure 7: Time as it varies with the number of constraints q .

Figure 8, again, shows that ELS-EIG and ELS-MAT are effective in reducing the number of SDP-CUT iterations needed, while BACKTRACKING fails to reduce the number of iterations needed. Figure 8 also shows a positive correlation between the number of SDP-CUT iterations (WAC-NEWTON* iterations) and n . From Figure 9, we see that the exact line search reduces the total time needed for our algorithm, especially as n gets larger. This is important, because as n grows, the number of SDP-CUT iterations and the time required per iteration both increase (see Figure 10). For small n , CONSTANT is quicker per SDP-CUT iteration, but as n grows there is no noticeable difference in the time per iteration between CONSTANT or either ELS implementation. This is because the computations needed for the optimization problem (3.4) becomes so small compared to the computations need to compute $\nabla \phi_w^*(x)$ and $\nabla^2 \phi_w^*(x)$.

4.3. Performance as m Varies

Here, we left the number of constraints constant at $q = 2$ and the number of variables fixed at $n = 2$. We varied the matrix sizes $m_j = m$ from 5 to 60. The number of SDP-CUT iterations (WAC-NEWTON* iterations) and run time were recorded for each problem. Below are plots

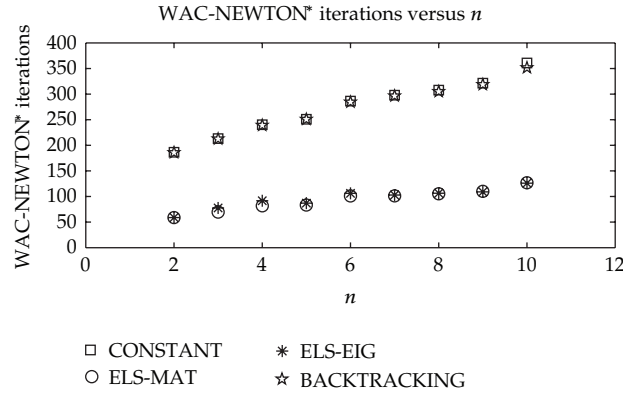


Figure 8: SDP-CUT iterations against the number of variables n .

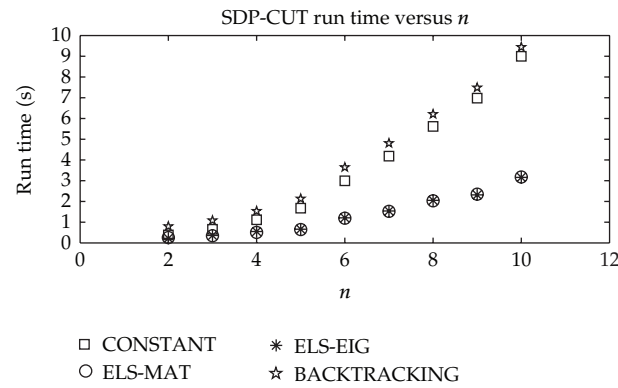


Figure 9: SDP-CUT time as it varies with the number of variables n .

of the averages with m on the horizontal axis. The vertical axis in Figure 11 contains the total SDP-CUT iterations (WAC-NEWTON* iterations) used to solve the problem. In Figure 12, the vertical axis gives the total run time of SDP-CUT in seconds. There is also a third plot, Figure 13, which shows m on the horizontal axis and the time required to perform an iteration of SDP-CUT on the vertical axis.

Figure 11, again, shows that ELS reduces the required number of SDP-CUT iterations (WAC NEWTON* iterations). However, as the size of the constraint matrices becomes large, the computations needed for the exact line search grow. Figure 12 shows the ELS-MAT is the fastest in the range of matrix sizes we tested. Unlike our other experiments, CONSTANT beats ELS-EIG. The reason for this is that the time per iteration of ELS-EIG grows very rapidly as is seen in Figure 13. This occurs because the computation of eigenvalues becomes increasingly difficult as the matrix grows in size.

ELS appears to be the best implementation of SDP-CUT. ELS is very effective in reducing the number of WAC NEWTON* iterations needed. ELS-MAT outperformed ELS-EIG in general. The eigenvalues allow for fast computation of the gradient and Hessian of $g(\alpha)$ once the eigenvalues are obtained (see (3.12) and (3.13)). However, the large cost of computing the eigenvalues outweighs this benefit, especially as the matrices become large. ELS-MAT uses a slower means of computing the gradient and Hessian of $g(\alpha)$, but has no

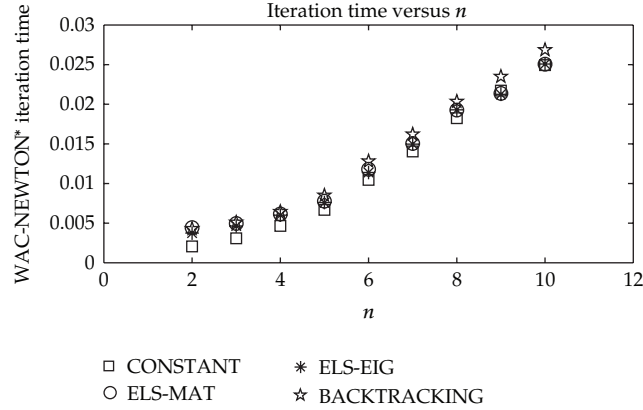


Figure 10: Time required to perform an iteration of SDP-CUT.

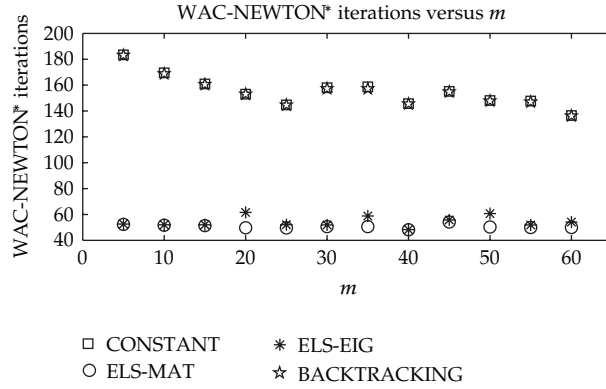


Figure 11: SDP-CUT iterations with matrix size.

eigenvalue calculation overhead (see (3.10) and (3.11)). The slowest operation needed is matrix inversion. Matrix inversion becomes harder as the matrices get large, but it scales much better than finding the eigenvalues. It is conceivable that very large matrix situations could arise in which both ELS-EIG and ELS-MAT are too expensive; in this case it may be best to use CONSTANT. We were unable to find any cases in our test problems in which BACKTRACKING was beneficial. From now onwards, SDP-CUT will always refer to SDP-CUT with ELS-MAT implementation.

4.4. Convergence and the Weight w , Leveling Off Effect

Increasing the weight w will decrease the number of SDP-CUT iterations needed to find the optimal solution. A larger weight will also allow us to get closer to the optimal solution. However, if w becomes too large, numerical errors will prevent convergence. Here, we will discuss the role w plays in the rate of convergence, as well as how large we can make w before SDP-CUT fails. Figure 4 graphically demonstrates the convergence of SDP-CUT on our example (SDP (2.3)) as the weight w varies.

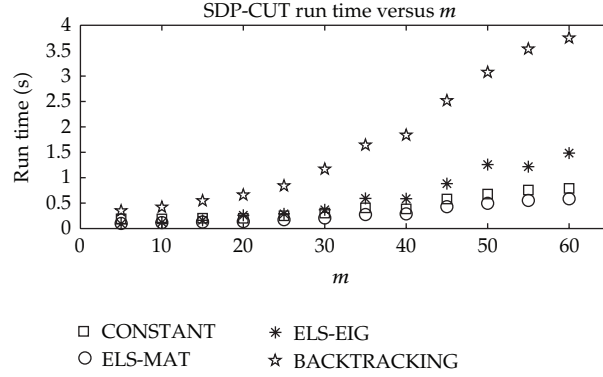


Figure 12: SDP-CUT time as it varies with matrix size.

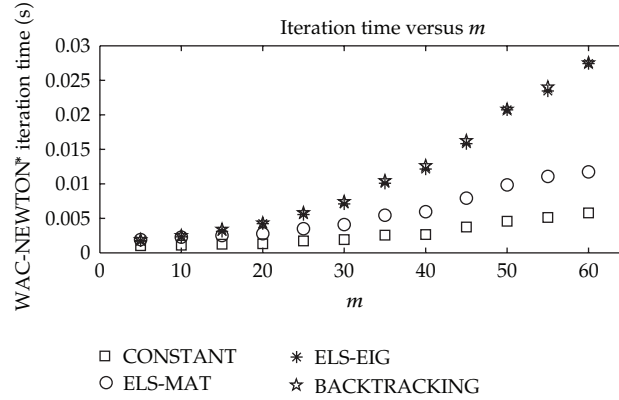
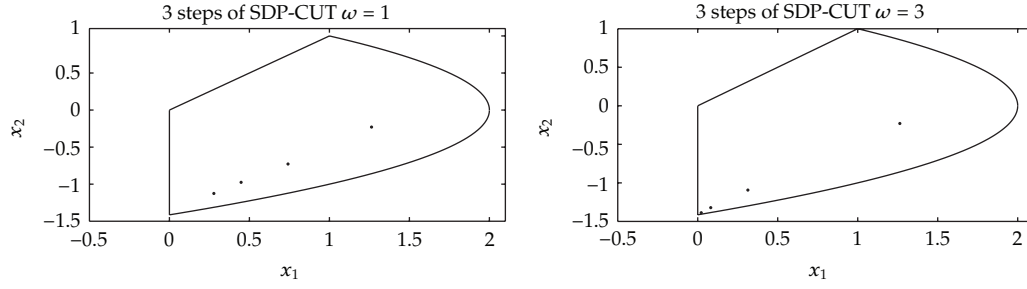
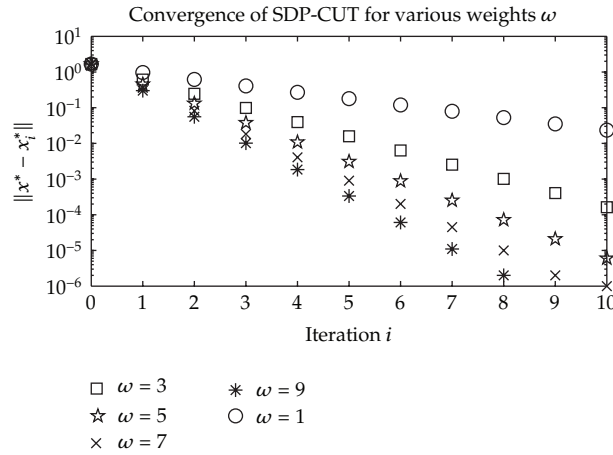


Figure 13: Time required to perform an iteration of WAC-NEWTON*.

From the plots in Figure 14, we can visibly see that by increasing the weight, SDP-CUT moves toward the optimal solution faster. In attempts to see how fast SDP-CUT converges and what effect the weight has, we performed the following experiment. We ran SDP-CUT on the example and kept track of the iterates x_k^* at each iteration. Below is a plot of $\|x^* - x_k^*\|$ shown on a log scale against iterations as we varied the weight $w = 1, 3, 5, 7, 9$, where x^* is the optimal solution. We see in Figure 15 that the distance from the estimate to the actual solution decreases exponentially, and linearly when shown on the log scale. We found that in most cases our implementation of SDP-CUT breaks down around $w = 10$.

As we can see, increasing the weight means quicker convergence. However, Figure 15 only shows the first 10 iterations. As we approach the optimal solution, the rate of convergence slows and “levels off.” This is caused by the feasible region becoming very small and by moving the cutting constraint back by ϵ , and thus the SDP-CUT iterates stay in approximately the same place. This problem is demonstrated in Figure 16. We see that larger weights allow us to get closer to the optimal solution, but even with a larger weight our convergence “levels off.”

One way to get closer to the optimal solution before the weighted analytic centers start to “level off,” is to use a smaller ϵ . The difficulty with decreasing ϵ is if ϵ becomes too small, our SDP-CUT iterates come too close to the boundary, giving rise to “numerical problems,”

Figure 14: Convergence of SDP-CUT for $w = 1$ and $w = 5$.Figure 15: Convergence for various weights w on a log scale.

especially in computing the Hessian matrices. We found that SDP-CUT worked with $\epsilon = 10^{-6}$, but failed when $\epsilon = 10^{-7}$ in example SDP (2.3). The “numerical problems” are due to the fact that near the boundary, one of the LMI constraint matrices is near-singular, and we need to compute matrix inverses. Figure 17 shows what the “leveling off” effect looks like over the feasible region of SDP problem (2.3). Notice that the sequence of iterates x_k^* , from SDP-CUT, reaches a limit before it reaches the boundary of the region.

5. Experiment II: SDP-CUT versus SDPT3

In this section, the algorithms SDP-CUT and the well-known SDPT3 [17] method are compared on a variety of SDP test problems. Since SDPT3 is known to be efficient, we also used it to find the best possible estimation of the actual optimal objective function values. Another reason for using SDPT3 is its flexibility to allow the user to input a starting point.

For the experiment, we tested 20 SDP problems and solved each with SDP-CUT and SDPT3. Each SDP problem was randomly generated, where the dimension n and number of constraints q are random integers on the intervals $[2, 20]$ and $[1, 20]$ respectively. The size m_j of each LMI is a random integer on $[1, 10]$. The values of n and q are listed in the table below. The values of m_j are not given because they are too many. For example, SDP 1 has $m = [7, 5, 9, 8, 5, 1]$. The problems were generated in a way that makes the origin, an interior point

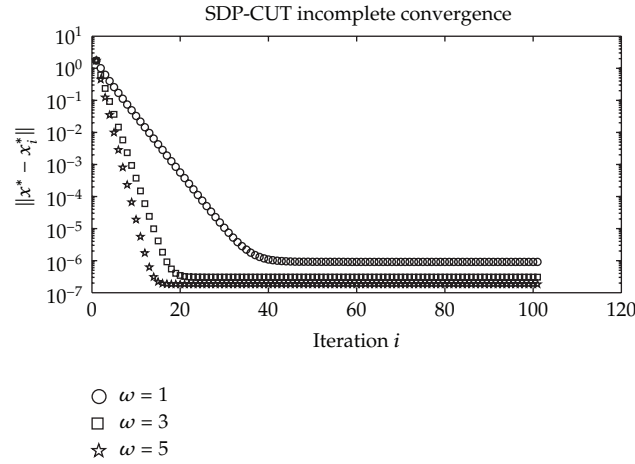


Figure 16: The “leveling off” effect.

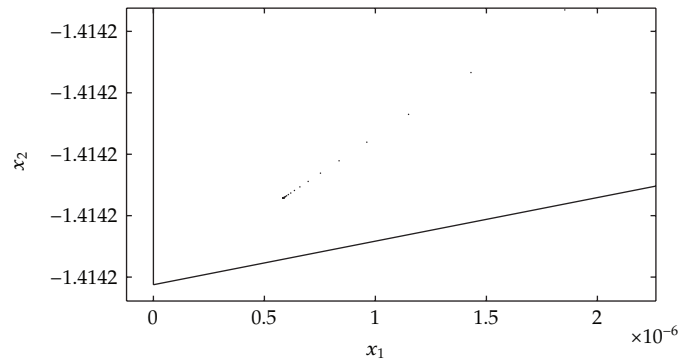


Figure 17: The “leveling off” effect zoomed in on feasible region.

for all of them. For all problems, we used tolerances $STOL = 10^{-3}$, $WTOL = 10^{-6}$, $\epsilon = 10^{-7}$, and a weight of $w = 7$ was used with SDP-CUT. For the first 10 test problems, the iteration limit was $MAX = 5$. For the remaining 10 problem, the iteration limit is raised to $MAX = 100$. We record the total iterations performed for each algorithm as well as the time taken. The last two columns of the table below show the error in the calculated optimal objective function value compared to the actual value p^*_{actual} obtained from SDPT3 with no restriction on the number of iterations. In the table below, the suffixes_{cut} and SDPT3 are used to distinguish between values pertaining to SDP-CUT and SDPT3, respectively. N , T , and E are used to denote the number of iterations, time, and absolute error, respectively. For example, $E_{\text{cut}} = |p^*_{\text{cut}} - p^*_{\text{actual}}|$.

We see in Table 1 that after 5 iterations, SDP-CUT gives a more accurate answer in all the first 10 test problems. However, if allowed to run for more iterations all the algorithms successfully found the optimal solutions, but SDPT3 took less number of iterations to reach optimality in 6 out of the 10 problems. Also SDPT3, took less time in all the 20 test problems. We believe the difference in times is partly because SDPT3 has over the years been optimized to run efficiently, while SDP-CUT is at a beginning of its development. It is interesting to note that SDP-CUT took fewer iterations in 4 out of the 10 problems.

Table 1: Data for experiment II: SDP-CUT versus SDPT3.

SDP	n	q	N_{cut}	N_{SDPT3}	T_{cut}	T_{SDPT3}	E_{cut}	E_{SDPT3}
1	20	6	5	5	2.6842	0.1134	0.0814	0.8222
2	12	17	5	5	3.2816	0.0954	0.0126	0.7291
3	18	15	5	5	5.0180	0.0905	0.1053	0.1361
4	13	2	5	5	0.5383	0.0732	0.0116	0.0721
5	2	5	5	5	0.0714	0.0600	0.0000	0.0011
6	4	7	5	5	0.2779	0.0814	0.0006	0.1159
7	3	19	5	5	0.4144	0.0766	0.0001	0.4938
8	15	5	5	5	1.5041	0.0969	0.0423	0.1689
9	5	2	5	5	0.1164	0.0603	0.0013	0.5072
10	20	12	5	5	5.2449	0.0906	0.1425	1.3682
11	4	2	10	12	0.1457	0.1324	0	0
12	4	3	8	12	0.1536	0.1322	0	0
13	16	5	20	13	4.8852	0.1791	0	0
14	18	9	25	14	12.1776	0.2330	0	0
15	20	15	22	12	19.7580	0.1971	0	0
16	15	15	19	12	10.4692	0.2117	0	0
17	10	8	14	12	2.2191	0.1759	0	0
18	13	7	16	11	3.4501	0.1647	0	0
19	2	20	8	11	0.4571	0.1552	0	0
20	5	18	9	10	1.0308	0.1747	0	0

We need to do more work to make SDP-CUT more efficient. We noticed that the Hessian matrix $\nabla^2 \phi_w^*(x)$ in SDP-CUT is prone to errors in the case of problems that are very large. In those cases, $\nabla^2 \phi_w^*(x)$ loses its positive definiteness and thus its Cholesky factorization is not possible. That is the main reason why we could not run SDP-CUT on the SDP test problems at SDPLIB [18]. For example, SDP-CUT was not successfully in solving *truss2* of SDPLIB, due to numerical problems when computing the Hessian matrices. This problem has $n = 58$ variables and $q = 133$ constraints. However, SDP-CUT was successful in solving *truss1*, but with $w = 5$.

6. Convergence of SDP-CUT

Let $\{x_k^*\}$ be the sequence of estimates for x^* generated by SDP-CUT. Let \mathcal{R}_k be the feasible region after k -cuts. Thus, x_k^* is the weighted analytic center of \mathcal{R}_k for all $k > 0$. Let $\{\partial_k\}$ be the sequence defined by $\partial_k = c^T(x_k^* - x_{k+1}^*)$.

The following theorem shows that when $\|c\|$ and $\|x_k^* - x_{k+1}^*\|$ are both small, then ∂_k is also small. Note that when ∂_k becomes very small, SDP-CUT is no longer making progress on the objective function values. This is especially true in the case of “leveling off” effect discussed in Section 4. This also indicates that our stopping criterion $\|x_k^* - x_{k+1}^*\| < \text{STOL}$ does not always mean convergence to the optimal solution.

Theorem 6.1. *If $\|x_k^* - x_{k+1}^*\| < \text{STOL}$, then $\partial_k < \|c\| \text{STOL}$.*

Proof. Assume $\|x_k^* - x_{k+1}^*\| < \text{STOL}$ and use the Cauchy-Schwartz Inequality

$$\begin{aligned}\partial_k &= c^T (x_k^* - x_{k+1}^*) \\ \implies \partial_k &\leq \|c\| \|x_k^* - x_{k+1}^*\| \\ \implies \partial_k &< \|c\| \text{STOL}.\end{aligned}\tag{6.1}$$

□

How close we can get to the optimal solution depends on the parameters w and ϵ . As ϵ becomes smaller we are able to obtain smaller regions, which allows SDP-CUT to get closer to the optimal solution. For a given region, a larger w causes SDP-CUT push the estimate of the optimal solution x_k^* further in the $-c$ direction. For any choice of iterate x_k^* , if we consider its limit as $\epsilon \rightarrow \infty$, we should be able to push x_k^* as close to the optimal solution x^* as we like. In practice, both the decrease $\epsilon \rightarrow 0$ and $w \rightarrow \infty$ cause numerical problems. A question we propose is: in the limit as $w \rightarrow \infty$, how many iterations of SDP-CUT are needed for convergence?

Since the cutting plane is a linear constraint, it is possible that only one iteration is needed as $w \rightarrow \infty$ (at least in some cases). Consider the following SDP (LP) example:

$$\begin{aligned}\text{SDP: minimize } & x_2 \\ \text{subject to } & 1 - x_1 \geq 0 \\ & 1 + x_1 \geq 0 \\ & 1 - x_2 \geq 0 \\ & 1 + x_2 \geq 0.\end{aligned}\tag{6.2}$$

The feasible region of this example is given in Figure 18. Consider the point $(0, 0)$ in the feasible region and the cutting constraint through this point. Also, consider the corresponding barrier function $\phi_w^*(x)$ (3.1) and its gradient $\nabla \phi_w^*(x)$ (3.2). At the weighted analytic center, the gradient is zero. This gives

$$\begin{aligned}\frac{1}{1 - x_1} - \frac{1}{1 + x_1} &= 0, \\ \frac{w}{\epsilon - x_2} + \frac{1}{1 - x_2} - \frac{1}{1 + x_2} &= 0.\end{aligned}\tag{6.3}$$

It is clear from Figure 18 that the weighted analytic center has a negative x_2 value. So, the above equations show that the weighted analytic center is

$$\begin{aligned}x_1 &= 0, \\ x_2 &= \frac{\epsilon - \sqrt{\epsilon^2 + 2w + w^2}}{2 + w}.\end{aligned}\tag{6.4}$$

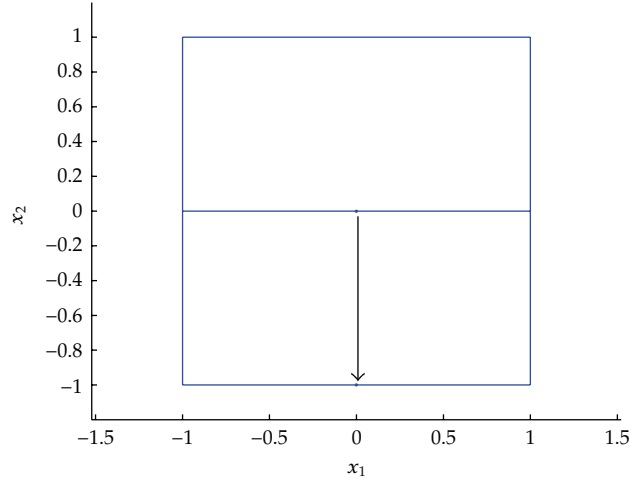


Figure 18: Feasible region of LP and the cut through the point $(0, 0)$.

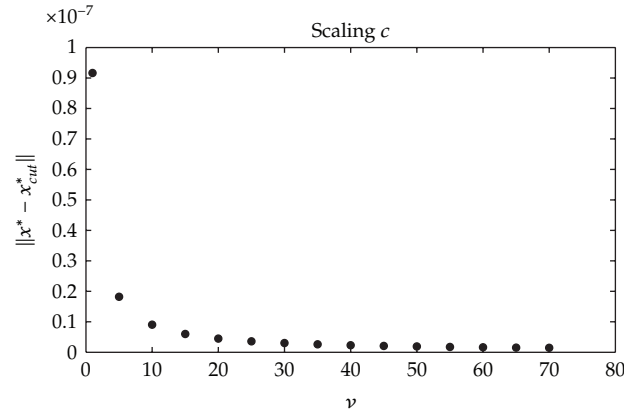


Figure 19: The effect of scaling c .

We see that the weighted analytic center (x_1, x_2) converges to the optimal solution $(0, -1)$ as $w \rightarrow \infty$. This example suggests the following conjecture.

Conjecture 6.2. *SDP-CUT converges to x^* as $w \rightarrow \infty$.*

6.1. Scaling the Vector c

Theorem 6.1 shows the dependence of ∂_k on $\|c\|$. Here, we will consider the effect of scaling c on SDP-CUT. If we scale c by $\nu > 1$ such that we are now trying to optimize $\nu c^T x$, we do not change the optimal solution x^* . We will compare how close SDP-CUT gets when optimizing for various values of ν . Figure 19 is a plot of the results $\|x^* - x_{\text{cut}}^*\|$ on the vertical axis and the scaling factor ν is on the horizontal axis. We used example SDP (2.3) with $\text{STOL} = 10^{-12}$, $\text{MAX} = 100$, $w = 10$. We can see in Figure 19 that scaling the objective vector c by a factor $\nu > 1$ does have a positive effect on the accuracy of SDP-CUT.

7. Conclusion

We have developed and studied a new interior point method, SDP-CUT, for solving SDPs. We found that implementing SDP-CUT using Newton's method with exact line search, in general, gives the best performance. SDP-CUT appears to be very effective in getting into the neighborhood of the optimal solution as seen in problems 1–10 in Table 1. However, there is the "leveling off" effect shown in Figure 16. Future work on SDP-CUT can involve investigating Conjecture 6.2. Implementations that can handle a larger weight w or smaller ϵ will result in better performance.

Another area for improvement is the stopping criteria. Currently, the stopping criteria is $\|x_k^* - x_{k-1}^*\| < \text{STOL}$, which lets us know if we are no longer making progress. However, this does not always guarantee optimality of the solution. We have also shown that scaling the objective vector c by some factor $\nu > 1$ could allow SDP-CUT to give a better solution. Also, it is important to investigate how to make all the other components of SDP-CUT work well and efficiently. For example, we would like to find a way to compute the Hessian $\nabla^2 \phi_w^*(x)_{ik}$ more efficiently and without errors. The way we compute the stepsizes in Newton's method is an area that requires further attention. The exact line search we preferred over the others is probably too expensive and a better way may be possible. These improvements would allow us to test SDP-CUT on larger problems and compare it with other methods. It is also of interest to study what kind of problems SDP-CUT would work well or fail.

Acknowledgment

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

References

- [1] S. Jibrin, *Redundancy in Semidefinite Programming: Detection and Elimination of Redundant Linear Matrix Inequalities*, VDM, Saarbrücken, Germany, 2009.
- [2] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Review*, vol. 38, no. 1, pp. 49–95, 1996.
- [3] L. Vandenberghe and S. Boyd, "Applications of semidefinite programming," *Applied Numerical Mathematics*, vol. 29, no. 3, 1999.
- [4] F. Alizadeh, "Interior point methods in semidefinite programming with applications to combinatorial optimization," *SIAM Journal on Optimization*, vol. 5, no. 1, pp. 13–51, 1995.
- [5] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, New York, NY, USA, 2004.
- [6] R. J. Caron, T. Traynor, and S. Jibrin, "Feasibility and constraint analysis of sets of linear matrix inequalities," *INFORMS Journal on Computing*, vol. 22, no. 1, pp. 144–153, 2010.
- [7] J. W. Chinneck, "The constraint consensus method for finding approximately feasible points in nonlinear programs," *INFORMS Journal on Computing*, vol. 16, no. 3, pp. 255–265, 2004.
- [8] W. Ibrahim and J. W. Chinneck, "Improving solver success in reaching feasibility for sets of nonlinear constraints," *Computers & Operations Research*, vol. 35, no. 5, pp. 1394–1411, 2008.
- [9] R. E. Gomory, "Outline of an algorithm for integer solutions to linear programs," *Bulletin of the American Mathematical Society*, vol. 64, no. 5, pp. 275–278, 1958.
- [10] J. E. Kelley, Jr., "The cutting-plane method for solving convex programs," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

- [11] E. W. Cheney and A. A. Goldstein, "Newton's method for convex programming and Tchebycheff approximation," *Numerische Mathematik*, vol. 1, no. 1, pp. 253–268, 1959.
- [12] J. Renegar, "A polynomial-time algorithm, based on Newton's method, for linear programming," *Mathematical Programming*, vol. 40, no. 1–3, pp. 59–93, 1988.
- [13] I. S. Pressman and S. Jibrin, "The weighted analytic center for linear matrix inequalities," *Journal of Inequalities in Pure and Applied Mathematics*, vol. 2, no. 3, article 29, 2002.
- [14] C. B. Chua, "A new notion of weighted centers for semidefinite programming," *SIAM Journal on Optimization*, vol. 16, no. 4, pp. 1092–1109, 2006.
- [15] S. Jibrin and J. W. Swift, "The boundary of weighted analytic centers for linear matrix inequalities," *Journal of Inequalities in Pure and Applied Mathematics*, vol. 5, no. 1, article 14, 2004.
- [16] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, New York, NY, USA, 2nd edition, 2006.
- [17] K. C. Tutuncu and M. J. Todd, *On the Implementation and Usage of SDPT3-a Matlab Software Package for Semidefinite-Quadratic-Linear Programming Version 4*, 2006.
- [18] B. Borchers and L. Vandenberghe, "SDPLIB 1.2, a library of semidefinite programming test problems," *Optimization Methods and Software*, vol. 11-12, no. 1–4, pp. 683–690, 1999.

