Extensions of Semidefinite Coordinate Direction Algorithm for Detecting Necessary Constraints to Unbounded Regions*

Susan Perrone

Department of Mathematics and Statistics

Northern Arizona University, Flagstaff, AZ 86011.

July 26, 2002

Abstract

In linear programming and semidefinite programming (SDP) it is highly beneficial, especially in problems with a large number of constraints, to eliminate redundant constraints or identify necessary constraints before solving the program. This process results in saved time and therefore lower costs, which is especially important for corporations who are often required to solve extensive optimization problems.

There already exist various methods for detecting redundant constraints in linear programming, but eliminating redundant constraints in (SDP) is an extremely hard problem. However, there are some practical methods from detecting necessary constraints in SDP, including the Semidefinite Stand-and-Hit, Semidefinite Hypersphere Direction, and Semidefinite Coordinate Direction methods. These methods work well except when trying to detect necessary constraints in unbounded feasible regions.

This paper examines four extensions of the Semidefinite Coordinate Direction method. These new methods work in both bounded and unbounded feasible regions and therefore are applicable to more linear and semidefinite programs.

*This research was made possible by the National Science Foundation Research Experience for Undergraduates Program and with the support of Dr. Shafiu Jibrin.

Contents

List of Figures

List of Tables

1 Background in Linear and Semidefinite Programming

Linear and *Semidefinite Programming* are optimization methods for systems of equalities and inequalities. Linear programming, a precursor to semidefinite programming, involves the maximization or minimization of an objective function subject to certain constraints. The constraints are linear equalities or inequalities that limit how much the variables can be increased or decreased. A sample linear program appears as follows:

$$\begin{aligned} min \quad & c^T x \\ s.t. \quad & b_j + a_j^T x \geq 0,\ j = 1 \dots q \end{aligned}$$

In the above program $x,\ c,\ b_j,\ a_j \in \mathbb{R}^n$.

The most commonly used methods for solving linear programs such as this are the *Simplex Method*, introduced by Dantzig in the 1940s, and the *Barrier-Interior Point Methods*, developed in the 1960s by Fiacco and McCormick and applied in the 1980s by Karmarkar [3]. The simplex method works by moving to successive intercepts (or corner points) of the feasible region, while the interior point methods begin by selecting a point in the interior of the feasible region and moving in the optimal direction to find the optimal solution to the objective function.

Linear programming is highly useful in modeling and solving problems in planning, routing, scheduling, assignments, and design and is used in various industries such as transportation, energy, telecommunications, and manufacturing [3].

In the 1990s, a new area called Semidefinite Programming (SDP) received a great deal of attention from leading researchers because of its powerful applications and interesting theory [9]. The applications are more powerful because the programs involve constraints

that are matrix inequalities, and allow for approximate solutions to NP-hard Combinatorial Optimization problems. A general SDP program is as follows:

$$\begin{aligned} min \quad & c^T x \\ s.t. \quad & A^{(j)}(x) := A_0^{(j)} + \sum_{i=1}^{n} x_i A_i^{(j)} \succeq 0, \ j = 1 \dots q \end{aligned}$$

In the above program $x \in \mathbb{R}^n$ and all matrices are symmetric with dimension m_j. The constraint $A^{(j)}(x) \succeq 0$ is called a linear matrix inequality (LMI). We write $F \succeq 0$ to mean that the matrix F is positive semidefinite. $F \succeq 0$ if an only if all eigenvalues of F are nonnegative. F is positive definite (written $F \succ 0$) if and only if all eigenvalues are strictly positive.

Semidefinite programming is still an active area of research in optimization. An understanding of this area will allow for broader range of problem solving abilities.

Because these problems are of interest to large industries, who solve a large number of these programs on a daily basis, it is valuable to research the methods for detecting and eliminating redundant constraints in semidefinite programming. A *necessary constraint* is one whose removal changes the feasible region, while the removal of a *redundant constraint* has no effect. The time it takes to solve a program increases as the number of constraints increases. Therefore, it is cost efficient to remove all of the redundant constraints before solving, especially in a large SDP where numerous unnecessary constraints may exist.

Research has been developed on detecting and eliminating these redundant constraints from linear programs. The first efforts were made in the 1960s, but recently there have been great advances in this study [6]. It is possible to determine which linear constraints are redundant by solving a linear program. Telgen developed an algorithm called REDINQ, which eliminates these redundant constraints [6]. First a tableau representing a basic feasible solution is examined to see for which constraints the conditions of his Theorems

2.1 through 2.6 hold [6]. These are eliminated as redundant and then the algorithm continues by selecting a unidentified constraint [6]. After performing a pivot operation of the Simplex Method, the constraint is identified as redundant or not and then the algorithm is repeated until all of the constraints have been categorized [6]. This method can be applied to both the primal and dual problems, and by combining the steps with the complementary slackness theorem, Telgen shows that the problem can be greatly reduced [6]. Several other methods for eliminating redundant linear constraints are given in [5].

Theorem 1.1 *Redundancy Theorem [4]*

$A^{(k)}(x) \succ 0$ *is redundant if and only if the semidefinite program*

$$SDP_k: \quad min \quad \lambda_{min}(A^{(k)}(x))$$

$$s.t. \quad x \in \mathcal{R}_k$$

has an optimal solution, say x^**, satisfying* $\lambda_{min}(A^{(k)}(x^*)) \geq 0$.

This theorem shows that detecting redundant linear matrix inequalities is a hard problem, due to the concave nature of the function. In [4], Monte Carlo methods were developed for detecting necessary linear matrix inequalities in bounded regions. These are the Semidefinite Stand-and-Hit (SSH), Semidefinite Hypersphere Direction (SHD), and Semidefinite Coordinate Direction methods (SCD).

2 Monte Carlo Methods for Detecting Necessary Constraints

In this section we describe the three Monte Carlo methods listed above, SSH, SHD, and SCD, as given in [4]. All of these methods involve selecting a *standing point* x_0 in the interior of the feasible region.

Definition 2.1 *A point* x *is in the interior of the feasible region if and only if* $A(x) \succ 0$. *The point* x *is on the boundary of the feasible region if and only if* $A(x) \succeq 0$ *and* $A(x) \not\succ 0$ *and* $\mid A(x) \mid = 0$.

The following shows how to find a point x^* in the interior of $A(x) \succeq 0$ that can be used as the standing point in the algorithm.

Theorem 2.1 *Interior Point Feasibility Theorem*

Given $A(x) \succeq 0$

$$\begin{aligned} min \quad & t \\ s.t. \quad & A(x) + tI \succeq 0 \end{aligned}$$

If (x^*, t^*) *is an optimal solution, then* $A(x^*) \succ 0$ *if and only if* $t^* < 0$.

Proof: Suppose $A(x^*) \succ 0$. Now, $A(x^*) + t^*I \succeq 0$ and $A(x^*) + t^*I \not\succ 0$. This implies that $\lambda_{min}(A(x^*) + t^*I) = 0$. Therefore we know that $\lambda_{min}(A(x^*)) + t^* = 0$ and so $t^* < 0$. Conversely, suppose $t^* < 0$. Since (x^*, t^*) is an optimal solution, $A(x^*) + t^*I \succeq 0$. This implies that $\lambda_{min}(A(x^*)+t^*I) \geq 0$ and $\lambda_{min}(A(x^*))+t^* \geq 0$. Therefore, $\lambda_{min}(A(x^*)) > 0$ and $A(x^*) \succ 0$.

2.1 Semidefinite Stand and Hit Method (SSH)

In the Stand and Hit method, after a *standing point* is found using the theorem shown above, a random search direction s is then selected and line segments $\{x_0+\sigma s \mid \sigma \geq 0\}$ and $\{x_0 - \sigma s \mid \sigma \geq 0\}$ are formed between the *standing point* and the *hit point*, which occurs where the line intersects the boundary of the feasible region. In order to determine the necessary constraints, the minimum distance σ from the standing point to the hit points is calculated and the necessary constraints are identified.

Definition 2.2 *Given a search direction vector* s*, define the symmetric matrix at* x_0

$$B_j(s, x_0) = -A^{(j)}(x_0)^{-1/2}(\sum_{i=1}^{n} s_i A_i^{(j)})A^{(j)}(x_0)^{-1/2} \text{ for all } j \in \hat{q}$$

SSH Algorithm

Initialization Denote the set of identified constraints by $\mathcal{J}$ and set $\mathcal{J} = \emptyset$. Choose an interior (standing) point x_0 of $\mathcal{R}$. Calculate $A^j(x_0)^{-1/2}$ for $f \in \hat{q}$.

Repeat

Search Direction: Choose $\tilde{s}$ with n entries from N(0,1) and compute $s = \tilde{s}/\| \tilde{s} \|_2$ to generate a random point s uniformly on the surface of $S^{n-1} = \{x \in \mathbb{R}^n \mid \| \tilde{x} \|_2 = 1\}$.

Hitting Step: Calculate $B_j(s, x_0)$ and find $\sigma_+^{(j)},\ \sigma_-^{(j)}$ for all $j \in \hat{q}$.

Calculate $\sigma_+ = \min\{\sigma_+^{(j)} \mid j \in \hat{q}\}$ and $\sigma_- = \min\{\sigma_-^{(j)} \mid j \in \hat{q}\}$.

For $1 \leq k \leq q$, if $\sigma_+^{(k)} = \sigma_+$ or $\sigma_-^{(k)} = \sigma_-$ and $k \notin \mathcal{J}$, set $\mathcal{J} = \mathcal{J} \cup \{k\}$.

Until a stopping rule holds.

2.2 Variations of SSH: Semidefinite Hypersphere Direction (SHD) and Semidefinite Coordinate Direction (SCD)

A variation of the SSH method is the Hypersphere Direction Method or SHD. In the previous algorithm, the *standing point* x_0 was fixed, limiting the identification of necessary constraints. In this method the standing point for each consecutive iteration is uniformly selected on the line segment between the previous standing point and hit points, improving the odds of locating all necessary constraints [4]. The differences between the two methods occurs in the hitting step, and the algorithm is as follows:

SHD Algorithm

Initialization Denote the set of identified constraints by $\mathcal{J}$ and set $\mathcal{J} = \emptyset$. Choose an interior (standing) point x_0 of $\mathcal{R}$. Calculate $A^j(x_0)^{-1/2}$ for $f \in \hat{q}$.

Repeat

Search Direction: Choose $\tilde{s}$ with n entries from N(0,1) and compute $s = \tilde{s}/\| \tilde{s} \|_2$ to generate a random point s uniformly on the surface of $S^{n-1} = \{x \in \mathbb{R}^n \mid\| \tilde{x} \|_2 = 1\}$.

Hitting Step: Calculate $B_j(s, x_0)$ and find $\sigma_+^{(j)},\ \sigma_-^{(j)}$ for all $j \in \hat{q}$.

Calculate $\sigma_+ = \min\{\sigma_+^{(j)} \mid j \in \hat{q}\}$ and $\sigma_- = \min\{\sigma_-^{(j)} \mid j \in \hat{q}\}$.

For $1 \leq k \leq q$, if $\sigma_+^{(k)} = \sigma_+$ or $\sigma_-^{(k)} = \sigma_-$ and $k \notin \mathcal{J}$, set $\mathcal{J} = \mathcal{J} \cup \{k\}$.

Choose u from U(0,1) and set $x_0 = x_0 + (u(\sigma_+ + \sigma_-) - \sigma_-)s$.

Until a stopping rule holds.

Another improvement of the SSH algorithm is called the Coordinate Direction Method or SCD. In this variation the standing point x_0 is not fixed, as in SHD, but the direction vector s is only selected in the coordinate directions [4]. This variation is advantageous because the direction vectors are selected more systematically. When the standing point x_0 is not fixed and the direction vector is limited to the coordinate directions, the algorithm is more likely to identify all necessary constraints.

Definition 2.3 *Given a search direction vector $s = e_i$, define the symmetric matrix at x_0*

$$B_j(s, x_0) = -A^j(x_0)^{-1/2}(\sum_{i=1}^{n} s_i A_i^{(j)})A^j(x_0)^{-\frac{1}{2}} = -A^j(x_0)^{-\frac{1}{2}}A_i^{(j)}A^j(x_0)^{-1/2} \text{ for all } j \in \hat{q}$$

SCD Algorithm

Initialization Denote the set of identified constraints by $\mathcal{J}$ and set $\mathcal{J} = \emptyset$. Choose an interior (standing) point x_0 of $\mathcal{R}$. Calculate $A^j(x_0)^{-1/2}$ for $f \in \hat{q}$.

Repeat

Search Direction: Choose $\tilde{s}$ randomly from the $2n$ directions $\pm e_i$ parallel to the coordinate axes and set $s = e_i$, where e_i is the canonical unit vector.

Hitting Step: Calculate $B_j(s, x_0)$ and find $\sigma_+^{(j)},\ \sigma_-^{(j)}$ for all $j \in \hat{q}$.

Calculate $\sigma_+ = \min\{\sigma_+^{(j)} \mid j \in \hat{q}\}$ and $\sigma_- = \min\{\sigma_-^{(j)} \mid j \in \hat{q}\}$.

For $1 \leq k \leq q$, if $\sigma_+^{(k)} = \sigma_+$ or $\sigma_-^{(k)} = \sigma_-$ and $k \notin \mathcal{J}$, set $\mathcal{J}$=$\mathcal{J} \cup \{k\}$.

Choose u from U(0,1) and set $x_0 = x_0 + (u(\sigma_+ + \sigma_-) - \sigma_-)$.

Until a stopping rule holds.

The following is a semidefinite program example.

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

$$A^{(2)}(x) = \begin{bmatrix} 3.8 & 0 \\ 0 & 3.8 \end{bmatrix} + x_1 \begin{bmatrix} -0.4 & 0 \\ 0 & -0.4 \end{bmatrix} + x_2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \succeq 0$$

$$A^{(3)}(x) = \begin{bmatrix} 2.6 & 0 \\ 0 & 2.6 \end{bmatrix} + x_1 \begin{bmatrix} 0.8 & 0 \\ 0 & 0.8 \end{bmatrix} + x_2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \succeq 0.$$

$$A^{(4)}(x) = \begin{bmatrix} 11.8 & 0 \\ 0 & 11.8 \end{bmatrix} + x_1 \begin{bmatrix} -2.0 & 0 \\ 0 & -2.0 \end{bmatrix} + x_2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \succeq 0.$$

Presently, the methods for detecting redundant constraints do not work properly in programs with unbounded feasible regions. In order to overcome this limitation, the developments in this paper will work with examples like the one above. The plot of the linear matrix inequalities can be seen in Figure 1.

The following sections are an examination of four extensions of the SCD algorithm. They will be implemented in both randomly produced linear matrix inequality constraints and in constraint systems, like the one above, that were manipulated to have unbounded feasible regions (see Table 4 in Appendix A for information about the tested SDPs).

3 Detecting Unboundedness using SCD

The research involving the use of these algorithms in unbounded feasible regions is somewhat limited. The known methods cannot be used because the algorithms cannot work when the distance σ is infinite. In order to eliminate these limitations the methods were

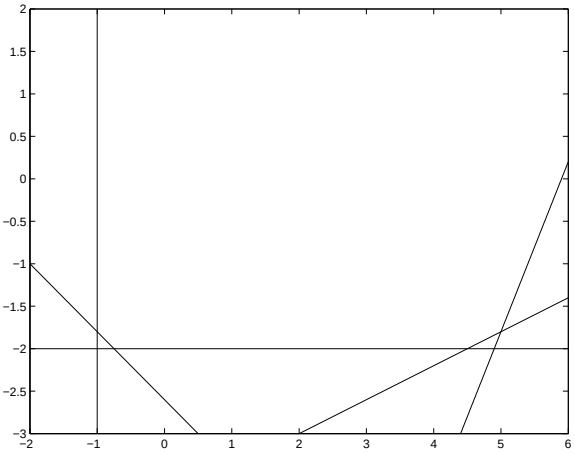

Figure 1: UBDATA 4

adapted to work in both bounded and unbounded regions, so that they could be implemented in all cases.

Theorem 3.1 *Unbounded Region Theorem*

The feasible region determined by $A^{(j)}(x) = A_0^{(j)} + \sum x_i A_i^{(j)} \succeq 0$ *is unbounded along the ray* $\{x_0 + \sigma s \mid \sigma \geq 0\}$ *if and only if the matrix* $B^{(j)}(x_0, s)$ *has no positive eigenvalues.*

Proof: Suppose $A^{(j)}(x) \succeq 0$ is unbounded along the ray $\{x_o + \sigma s \mid \sigma \geq 0\}$. Then

$$\begin{aligned} min \quad & \sigma \\ s.t. \quad & \mid A^{(j)}(x_0 + \sigma s) \mid = 0 \\ & \sigma > 0 \end{aligned} \tag{1}$$

is an infeasible problem. It suffices to show that if σ is a feasible point of problem, then σ is a positive eigenvalue of $B^{(j)}(x_0, s)$. Suppose σ is a feasible point of problem 1. Then

$$\mid A^{(j)}(x_0 + \sigma s) \mid = 0 \iff \mid A^{(j)}(x_0) + \sigma \textstyle\sum_{i=1}^{n} s_i A_i^{(j)} \mid = 0 \iff$$

$$\mid I + \sigma A^{(j)}(x_0^{(-1/2)})(\textstyle\sum_{i=1}^{n} s_i A_i^{(j)}) A^{(j)}(x_0^{(-1/2)}) \mid = 0 \iff$$

$$\mid 1/\sigma - B^{(j)}(x_0, s) \mid = 0 \iff 1/\sigma \text{ is a positive eigenvalue of } B^{(j)}(x_0, s)$$

Conversely, suppose α is the maximum positive eigenvalue of $B^{(j)}(x_0, s)$. Then $\sigma = 1/\alpha$ is an optimal solution of the above problem 1. Hence, $A^{(j)}(x) \succeq 0$ is bounded along the ray $\{x_0 + \sigma s \mid \sigma \geq 0\}$.

The theorem above demonstrates the technique for determining when a feasible region is unbounded in the s direction. Using the knowledge that $B^{(j)}(x_0, s)$ has no positive eigenvalues, we know the maximum eigenvalue in the positive direction will be zero. This is implemented in the following Method 1, Method 3, SCDVar, and SCDVar-SHD algorithms.

4 Previous Methods for Detecting Redundant Constraints in Unbounded Regions

In past years some ideas have been developed but not implemented in the semidefinite programs. These ideas are titled Method 1 and Method 3 and are described in the continuation of this section.

4.1 Method 1

In Method 1, the SCD algorithm is varied so that when unboundedness is detected, a new search direction is used with the same point as in the previous iteration [2]. The hitting step will be as follows:

Hitting Step: Calculate $B_j(s, x_0)$ and find $\sigma_+^{(j)},\ \sigma_-^{(j)}$ for all $j \in \hat{q}$.

Calculate $\sigma_+ = \min\{\sigma_+^{(j)} \mid j \in \hat{q}\}$ and $\sigma_- = \min\{\sigma_-^{(j)} \mid j \in \hat{q}\}$.

For $1 \leq k \leq q$, if $\sigma_+^{(k)} = \sigma_+$ and $\sigma_+^{(k)} \neq 0$ or $\sigma_-^{(k)} = \sigma_-$ and $\sigma_-^{(k)} \neq 0$, if $k \notin \mathcal{J}$ then set $\mathcal{J}$=$\mathcal{J} \cup \{k\}$.

If $\sigma_+^{(k)} = 0$ or $\sigma_-^{(k)} = 0$ set $x_0 = x_0$.

Otherwise, choose u from U(0,1) and set $x_0 = x_0 + (u(\sigma_+ + \sigma_-) - \sigma_-)$.

4.2 Method 3

In Method 3, the SCD algorithm is varied so that when unboundedness is detected a new search direction is used with a new standing point. This new standing point is selected from a uniform distribution on the line between the previous standing point and the hit point that was found in the bounded direction [2]. The hitting step will be as follows:

Hitting Step: Calculate $B_j(s, x_0)$ and find $\sigma_+^{(j)},\ \sigma_-^{(j)}$ for all $j \in \hat{q}$.

Calculate $\sigma_+ = \min\{\sigma_+^{(j)} \mid j \in \hat{q}\}$ and $\sigma_- = \min\{\sigma_-^{(j)} \mid j \in \hat{q}\}$.

For $1 \leq k \leq q$, if $\sigma_+^{(k)} = \sigma_+$ and $\sigma_+^{(k)} \neq 0$ or $\sigma_-^{(k)} = \sigma_-$ and $\sigma_-^{(k)} \neq 0$, if $k \notin \mathcal{J}$ then set $\mathcal{J}$=$\mathcal{J} \cup \{k\}$.

Choose u from U(0,1)

If $\sigma_+^{(k)} = 0$ set $x_0 = x_0 + (u(\sigma_-) - \sigma_-)$.

If $\sigma_-^{(k)} = 0$ set $x_0 = x_0 + (u(-\sigma_+) + \sigma_+)$.

Otherwise, set $x_0 = x_0 + (u(\sigma_+ + \sigma_-) - \sigma_-)$.

4.3 Comparison of Methods

Theoretically, we expect that Method 1 will work better than Method 3. The algorithm for Method 1 is more efficient because the standing points move throughout the feasible region. When unboundedness is detected only the direction vector changes, and so the points are not limited towards the bounded region as in Method 3. The latter method will work efficiently only if the initial standing point x_0 is selected far from the bounded direction. Even in these cases, if there is a necessary constraint far in the unbounded direction Method 3 will have a smaller chance of detecting it.

The data shown in Table 1 proves that in most situations, Method 1 works faster than Method 3, and Method 1 is better at detecting all of the necessary constraints.

Method 3 also has limitations in that it does not work properly with a high number of iterations for certain unbounded[1] feasible regions. For example, with the experimentation of UBData the last necessary constraint was found sooner with Method 3 than Method 1, but only because Method 3 did not detect as many constraints.

Data Set	Time (*sec*)		Necessary Constraints		Unbounded	
	Method 1	Method 3	Method 1	Method 3	Method 1	Method 3
Rand1	0.1500	0.3400	4,5,6,7	4,5,6,7	Possibly	Possibly
Rand2	0.6010	0.2310	1,2,3,5	1,2,5	Possibly	Possibly
Rand3	0.2010	0.2100	1,3	1,3	Possibly	Possibly
Rand4	1.1620	0.2600	1,2,3	1,2,3	Yes	Yes
Rand5	0.7110	2.5540	1,2,3,4,5,6	1,2,3,4,5,6	Possibly	Possibly
Rand6	0.4510	0.5400	1,2,3,4	1,2,3,4	Possibly	Possibly
Rand7	0.7110	2.0230	2,3,4,5,6,7,8,9	2,3,4,5,6,7,8,9	Possibly	Possibly
UBData	0.8310	0.2100	1,2,3	1,2	Yes	Yes
UBData3	1.2820	1.9330	1,3,4,5	1,3,4,5	Possibly	Possibly
UBData4	1.4120	0.1500	1,2,3,4	1,2,3,4	Yes	Yes
Avg. Time	0.7512	0.8451				

Table 1: Comparison of Methods 1 and 3

The next section examines other possible ideas for handling the unbounded situations.

[1]The algorithm is set up to find whether or not the feasible region is unbounded. Depending on the shape of the region (see Figure 4 (Left)), unboundedness may not be detected. Not enough information is provided to say for certain that the region is bounded, but if unboundedness is detected, we know the region is unbounded.

5 New Methods for Detecting Redundant Constraints in Unbounded Regions

Due to limitations with the previously discussed algorithms, other ideas had to be developed. With Method 3, when the problem detects unboundedness the next standing point occurs in the bounded direction. For most cases this idea works, but if there is a necessary constraint far in the unbounded direction, it most likely would not be detected. Method 1 works well, but depending on the feasible region it could also miss some necessary constraints because the direction vectors are limited to only the coordinate directions.

For these reasons, the methods discussed in the continuation of this section were developed and tested to improve results.

5.1 SCD Variation (SCDVar)

This new variation of the SCD algorithm will allow for a new standing point to be chosen uniformly along the line found in any iteration. The difference between this concept and that of Method 3 is that the new standing point can be in the bounded or unbounded direction. This is useful because many redundant constraints could exist in the unbounded direction. In order to achieve this method, the hitting step of the SCD algorithm is as follows:

Hitting Step: Calculate $B_j(s, x_0)$ and find $\sigma_+^{(j)},\ \sigma_-^{(j)}$ for all $j \in \hat{q}$.

Calculate $\sigma_+ = \min\{\sigma_+^{(j)} \mid j \in \hat{q}\}$ and $\sigma_- = \min\{\sigma_-^{(j)} \mid j \in \hat{q}\}$.

For $1 \leq k \leq q$, if $\sigma_+^{(k)} = \sigma_+$ and $\sigma_+^{(k)} \neq 0$ or $\sigma_-^{(k)} = \sigma_-$ and $\sigma_-^{(k)} \neq 0$, if $k \notin \mathcal{J}$ then set $\mathcal{J}$=$\mathcal{J} \cup \{k\}$.

Choose u from U(0,1) and set $\sigma = -ln(u)$

If $\sigma_+^{(k)} = 0$ set $x_0 = x_0 + \sigma_- * s + \sigma * s$.

If $\sigma_-^{(k)} = 0$ set $x_0 = x_0 + \sigma_+ * s + \sigma * (-s)$.

Otherwise, choose u from U(0,1) and set $x_0 = x_0 + (u(\sigma_+ + \sigma_-) - \sigma_-)$.

To obtain a standing point in either direction, it was necessary to produce a random number σ, which is the negative natural log distribution of $x \in (0, 1)$ i.e. $\sigma = -ln\ x$ (see Figure 2). Because of this distribution, σ will usually be smaller than the distance between the standing point and the hit point. When unboundedness is detected and this occurs, a new standing point will be chosen in the bounded direction. But when σ is randomly chosen to be larger than the distance σ_+ or σ_-, the new standing point will be in the unbounded direction and the alrogithm can search for those necessary constraints which might be far away from the other constraints.

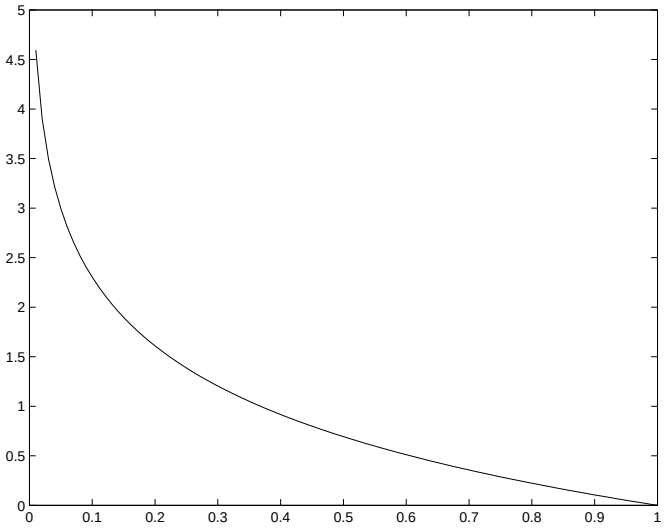

Figure 2: Negative Natural Log Distribution where $x \in (0, 1)$

This algorithm takes care of the limitations of Method 3, but by using only SCD the direction vectors, as in Method 1, are still limited to the coordinate directions.

5.2 Combination of SCD Variation and SHD (SCDVar-SHD)

In this final method, the SCD variation, with its ability to select a new standing point between the previous standing point and the unbounded direction, is combined with the SHD the algorithm, which is able to reach many more points than in previous methods by having vectors s in all directions.

In this algorithm, when unboundedness is detected a new standing point is found in the same way as was discussed in the previous method. If the point selected is between the previous standing point and the previous hit point (i.e. in the bounded direction), then the algorithm continues with SCD variation because there is no need for having direction vectors not in the coordinate directions. But if the point is between the previous standing point and the unbounded direction, the algorithm switches to SHD so that it can search in various directions. The changes to the algorithm are as follows:

Hitting Step: Calculate $B_j(s, x_0)$ and find $\sigma_+^{(j)},\ \sigma_-^{(j)}$ for all $j \in \hat{q}$.

Calculate $\sigma_+ = \min\{\sigma_+^{(j)} \mid j \in \hat{q}\}$ and $\sigma_- = \min\{\sigma_-^{(j)} \mid j \in \hat{q}\}$.

For $1 \leq k \leq q$, if $\sigma_+^{(k)} = \sigma_+$ and $\sigma_+^{(k)} \neq 0$ or $\sigma_-^{(k)} = \sigma_-$ and $\sigma_-^{(k)} \neq 0$, if $k \notin \mathcal{J}$ then set $\mathcal{J}$=$\mathcal{J} \cup \{k\}$.

Choose u from U(0,1) and set $\sigma = -ln(u)$

If $\sigma_+^{(k)} = 0$ and $\mid \sigma_-^{(k)} \mid \geq \sigma$ repeat SCD variation algorithm setting

$x_0 = x_0 + \sigma_- * s + \sigma * s$.

If $\sigma_+^{(k)} = 0$ and $\mid \sigma_-^{(k)} \mid < \sigma$ switch to SHD algorithm setting $x_0 = x_0 + \sigma_- * s + \sigma * s$.

If $\sigma_-^{(k)} = 0$ and $\sigma_+^{(k)} \geq \sigma$ repeat SCD variation algorithm setting

$x_0 = x_0 + \sigma_+ * s + \sigma * (-s)$.

If $\sigma_-^{(k)} = 0$ and $\sigma_+^{(k)} < \sigma$ switch to SHD algorithm setting $x_0 = x_0 + \sigma_+ * s + \sigma * (-s)$.

Otherwise, choose u from U(0,1) and set $x_0 = x_0 + (u(\sigma_+ + \sigma_-) - \sigma_-)$.

This method is highly useful because it is a combination all of the beneficial parts of the previously discussed methods.

5.3 Comparison of Methods

It can be said that SCDVar is only a building block of the second algorithm. SCDVar-SHD is just the SCDVar algorithm with the added advantage of randomly producing direction vectors in all directions, which suggests in theory that SCDVar-SHD should be the more efficient of the two.

In Table 2 we see that the theory is supported by the data, and the SCDVar-SHD method works more efficiently overall than the SCDVar method. They equally detect necessary constraints, but SCDVar-SHD has a slightly smaller average time and is also better at detecting unboundedness, as seen in the Rand3 experimentation.

Data Set	Time *(sec)*		Necessary Constraints		Unbounded	
	SCDVar	SCDVar-SHD	SCDVar	SCDVar-SHD	SCDVar	SCDVar-SHD
Rand1	1.2120	0.5210	4,5,6,7	4,5,6,7	Possibly	Possibly
Rand2	0.2610	0.3210	1,2,3,5	1,2,3,5	Possibly	Possibly
Rand3	0.2000	0.0900	1,3	1,3	Possibly	Yes
Rand4	0.2610	0.1410	1,2,3	1,2,3	Yes	Yes
Rand5	1.5220	1.6520	1,2,3,4,5,6	1,2,3,4,5,6	Possibly	Possibly
Rand6	0.5310	0.8210	1,2,3,4	1,2,3,4	Possibly	Possibly
Rand7	1.5620	1.4920	2,3,4,5,6,7,8,9	2,3,4,5,6,7,8,9	Possibly	Possibly
UBData	0.3310	0.1000	1,2,3	1,2,3	Yes	Yes
UBData3	2.4340	2.5800	1,3,4,5	1,3,4,5	Possibly	Possibly
UBData4	0.4000	0.3110	1,2,3,4	1,2,3,4	Yes	Yes
Avg. Time	0.8714	0.8029				

Table 2: Comparison of SCDVar and SCDVar-SHD

6 Drawing Boundaries of Feasible Regions using SCD Algorithms

Often, one would like to know what the feasible region looks like, or in what direction the program is unbounded. These algorithms can be altered slightly to add a highly beneficial plotting step. As seen in Figures 3, 4, and 5, a picture of the feasible region can be produced by plotting the hit points found along the boundary in every iteration. In Figure 6 we see that even the picture of a three dimensional feasible region can be produced.

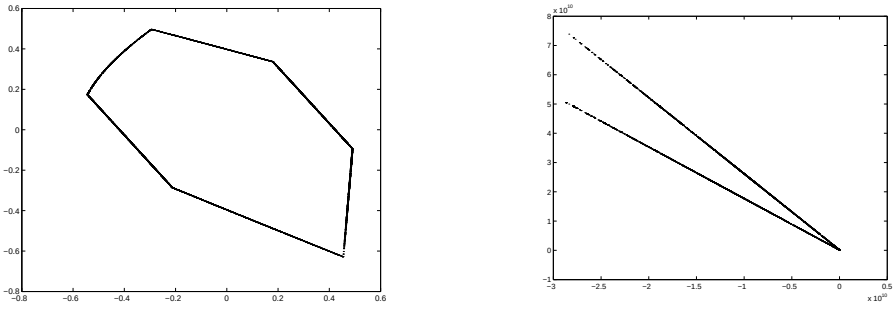

Figure 3: RAND 1 Boundary (Left) and RAND 2 Boundary (Right)

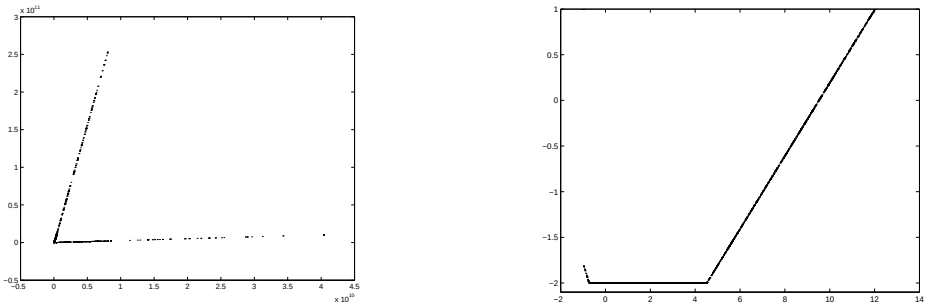

Figure 4: RAND 3 Boundary (Left) and UBDATA Boundary (Right)

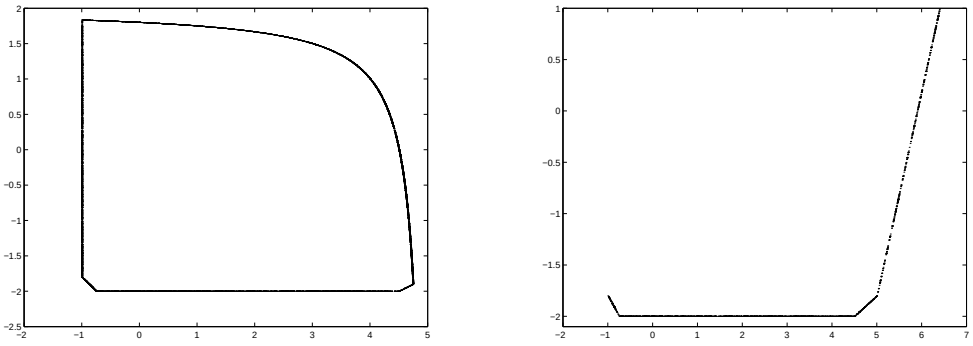

Figure 5: UBDATA 3 Boundary (Left) and UBDATA 4 Boundary (Right)

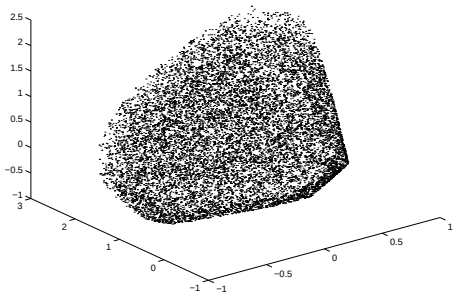

Figure 6: RAND 5 Three Dimensional Boundary

With this simple addition to the algorithm, the experiment shows the necessary constraints and also gives a visual of what is taking place in the semidefinite program.

7 Conclusions

Before deciding which methods are best, a decision has to be made regarding the importance of time or necessary constraints. For large corporations, time is a crucial factor in increasing profits and its minimization is the primary goal. But for our objective, handling all unbounded situations and maximizing the chance of detecting all necessary constraints is most important, so some sacrifice can be made in speed of the algorithms. Fortunately, the faster methods also identified a greater number of necessary constraints. In Table 3, Method 1 and SCDVar-SHD are compared.

Data Set	Time *(sec)*		Necessary Constraints		Unbounded	
	Method 1	SCDVar-SHD	Method 1	SCDVar-SHD	Method 1	SCDVar-SHD
Rand1	0.1500	0.5210	4,5,6,7	4,5,6,7	Possibly	Possibly
Rand2	0.6010	0.3210	1,2,3,5	1,2,3,5	Possibly	Possibly
Rand3	0.2010	0.0900	1,3	1,3	Possibly	Yes
Rand4	1.1620	0.1410	1,2,3	1,2,3	Yes	Yes
Rand5	0.7110	1.6520	1,2,3,4,5,6	1,2,3,4,5,6	Possibly	Possibly
Rand6	0.4510	0.8210	1,2,3,4	1,2,3,4	Possibly	Possibly
Rand7	0.7110	1.4920	2,3,4,5,6,7,8,9	2,3,4,5,6,7,8,9	Possibly	Possibly
UBData	0.8310	0.1000	1,2,3	1,2,3	Yes	Yes
UBData3	1.2820	2.5800	1,3,4,5	1,3,4,5	Possibly	Possibly
UBData4	1.4120	0.3110	1,2,3,4	1,2,3,4	Yes	Yes
Avg. Time	0.7512	0.8029				

Table 3: Comparison of Method 1 and SCDVar-SHD

As expected, we see from the data that SCDVar-SHD is the better of these two more efficient methods. SCDVar-SHD is the faster of the two algorithms 50% of the time, and

the average time is only slightly larger. SCDVar-SHD proves to be the better overall algorithm because, as seen in the Rand3 test, it is better at detecting unboundedness in certain situations.

There are numerous questions left unanswered thus far that can be examined with future research. We can see in Figures 3, 4, 5, and 6 that the points detected in every iteration occur around the entire boundary of the feasible region, one interesting question is whether or not there is a uniform distribution of the location of these points. If not, is there a way to generate a uniform distribution? We know that there can the distribution cannot be uniform for unbounded regions because of the infinite distance along certain constraints. The methods discussed in this paper can be used to first determine whether a feasible region is unbounded. Further work can be done to improve the algorithm so that we know for sure when a region is bounded as well, and then one could test how the location of hit points differs for bounded and unbounded problems. Instead of examining a distribution around the boundary, one could determine an algorithm for determining the maximum number of iterations needed to find all of the necessary constraints; is there any way to estimate this number?

The purpose behind the work that has been done is to decrease the amount of time necessary for solving, so perhaps the most useful research would be in the implementation of the above methods for solving optimization problems. If the algorithms are improved to tell the direction of unboundedness, that direction could then be compared to the direction of optimization. If we find that the two directions are similar, we would eliminate the need to solve the problem entirely because we would know that the optimization is unbounded and no solution exists.

A Appendix

The following table gives specifics about each of the examples used for implementing the four new methods. The linear matrix inequalities were produced randomly and so the SDPs randomly have bounded and unbounded feasible regions.

Data Set	Matrix Size	Number of Variables	Number of Constraints
Rand1	2	2	7
Rand2	1	2	5
Rand3	1	2	3
Rand4	1	3	3
Rand5	2	3	6
Rand6	2	5	4
Rand7	3	4	9
UBData	2	2	3
UBData3	2	2	5
UBData4	2	2	4

Table 4: Information on Tested SDPs

References

[1] R. Caron, A. Boneh and S. Boneh. "Redundancy". *Advances in Sensitivity Analysis and Parametric Programming*, eds. T. Gal and H. J. Greenberg, Kluwer Academic Publishers, International Series in Operations Research and Management Science, Vol. 6, 1997.

[2] S. Formaneck. "Improving the Semidefinite Coordinate Direction (SCD) Method for the Detection of Necessary Linear Matrix Inequalities". Northern Arizona University. Flagstaff, AZ, July 2001.

[3] R. Fourer. "Linear Programming Frequently Asked Questions". Optimization Technology Center of Northwestern University and Argonne National Laboratory. http://www-unix.mcs.anl.gov/otc/Guide/faq/ linear-programming-faq.html, 2000.

[4] S. Jibrin and Irwin S. Pressman. "Monte Carlo Algorithms for the Detection of Necessary Linear Matrix Inequality Constraints". Northern Arizona University. Flagstaff, AZ, September 2000.

[5] M. H. Karwan, V. Lofti, J. Telgen and S. Zionts. "Redundancy in Mathematical Pogramming". Springer-Verlag, Berlin, 1983.

[6] J. Telgen. "Identifying Redundant *Management Science*, Vol. 29, No. 10, October 1983.

[7] J. Telgen. "Redundancy and Linear Programs". Mathematical Centre Tracts, Amsterdam, 1981.

[8] L. Vandenberghe and S. Boyd. "Semidefinite Programming". *SIAM Review*, March 1996: 1-50.

[9] H. Wolkowicz. "Simple Efficient Solutions for Semidefinite Programming." University of Waterloo, Waterloo, Ontario, Canada, October 2001.