Hindawi Publishing Corporation
Journal of Optimization
Volume 2015, Article ID 790451, 16 pages
http://dx.doi.org/10.1155/2015/790451

Research Article

Constraint Consensus Methods for Finding Strictly Feasible Points of Linear Matrix Inequalities

Shafiu Jibrin and James W. Swift

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

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

Received 25 July 2014; Revised 31 October 2014; Accepted 6 November 2014

Academic Editor: Manlio Gaudioso

We give algorithms for solving the strict feasibility problem for linear matrix inequalities. These algorithms are based on John Chinneck's constraint consensus methods, in particular, the method of his original paper and the modified DBmax constraint consensus method from his paper with Ibrahim. Our algorithms start with one of these methods as "Phase 1." Constraint consensus methods work for any differentiable constraints, but we take advantage of the structure of linear matrix inequalities. In particular, for linear matrix inequalities, the crossing points of each constraint boundary with the consensus ray can be calculated. In this way we check for strictly feasible points in "Phase 2" of our algorithms. We present four different algorithms, depending on whether the original (basic) or DBmax constraint consensus vector is used in Phase 1 and, independently, in Phase 2. We present results of numerical experiments that compare the four algorithms. The evidence suggests that one of our algorithms is the best, although none of them are guaranteed to find a strictly feasible point after a given number of iterations. We also give results of numerical experiments indicating that our best method compares favorably to a new variant of the method of alternating projections.

1. Introduction

We consider the strict feasibility problem for linear matrix inequality (LMI) constraints. In particular, we seek a point $x \in \mathbb{R}^n$ in the interior of the region defined by q LMI constraints of the form

$$A^{(j)}(x) := A_0^{(j)} + \sum_{i=1}^{n} x_i A_i^{(j)} \succeq 0, \quad j \in J := \{1, 2, \ldots, q\}, \quad (1)$$

where $A_i^{(j)}$ are $m_j \times m_j$ symmetric matrices. The partial ordering $A \succeq B$ means $A - B$ is positive semidefinite. In semidefinite programming problems, a function is optimized over a system of LMI constraints. For more information on semidefinite programming including applications, refer to [1, 2]. Consider the feasible region

$$\mathscr{R} = \left\{x \in \mathbb{R}^n \mid A^{(j)}(x) \succeq 0, j \in J\right\}. \quad (2)$$

We assume that $\mathscr{R}$ has a nonempty interior. In this paper, we are interested in finding *strictly feasible* points, those in the interior of $\mathscr{R}$. Finding strictly feasible points is an important problem in interior point methods [3, 4]. Some methods need a starting point in the interior to proceed to optimality.

A projective method for solving the strict feasibility problem in the case of linear matrix inequalities (LMI's) was proposed in [5]. Their method is based on the Dikin ellipsoids and successive projections onto a linear subspace. The cost per iteration in this approach is high because each iteration solves a least-squares problem. A different approach based on the method of alternating projections was given in [6]. The iterations in this method are also costly as they require eigenvalue-eigenvector decompositions. The approach described in [7] solves the LMI feasibility problem by computing a minimum-volume ellipsoid at each iteration. The convergence rate is slow, so this method is also expensive. In [8], several iterative methods were given for the convex feasibility problem. These techniques use an orthogonal or a subgradient projection at each iteration.

We study the original (basic) constraint consensus method and the DBmax constraint consensus method developed by Chinneck [3] and Ibrahim and Chinneck [9] for

general nonlinear constraints to find near-feasible points. These methods also use gradient projection at each iteration to find a consensus vector that moves a point iteratively towards the feasible region starting from an infeasible point. The main cost in these methods is computing gradients, so they are relatively cheaper than the methods described in [5–7]. While the goal in [3, 9] is finding near-feasible points, ours is finding interior (strictly) feasible points. We apply and combine these methods to handle LMI constraints and find interior feasible points of $\mathscr{R}$ in two phases. Phase 1 is simply the original constraint consensus method or the DBmax constraint consensus method to find a near-feasible point.

In Phase 2, starting with the near-feasible point found in Phase 1, the algorithms use the concepts of crossing points and binary words to obtain a point in a most-satisfied interval along the ray of the consensus vector at each iteration. LMI constraints have the advantage that their crossing points are computable, whereas this is not possible for general nonlinear constraints. The goal in Phase 2 is to find an interior point of $\mathscr{R}$. We give four variations of the new algorithms: *Original-DBmax (OD) constraint consensus method*, *DBmax-DBmax (DD) constraint consensus method*, *Original-Original (OO) constraint consensus method*, and *DBmax-Original (DO) constraint consensus method*. The OD constraint consensus method uses the original constraint consensus method in Phase 1 and the DBmax search directions in Phase 2. The description of the other three methods is similar.

The paper is organized as follows. Section 2 gives a review of two of the constraint consensus methods introduced in [3, 9] and the computation of crossing points for LMIs. In Section 3 we apply the constraint consensus methods to a single LMI and present some theoretical results. For example, we provide necessary and sufficient conditions for the feasibility vector of a constraint to move the point to the boundary of that constraint. In Section 4 we give our four algorithms. Section 5 tests our algorithms on known benchmark problems. We also compare the four methods on a variety of randomly generated test problems. The results show that DO constraint consensus method outperforms our three other algorithms; it takes fewer total iterations and less computing time and has the most success in finding strictly feasible points. We compared DO with the method of alternating projections and found DO to outperform it on average, especially on problems with large number of LMI constraints relative to the number of variables. The concluding section has comments on how our algorithms could be improved and extended. While we focus on LMI constraints, our methods are applicable to other constraints, including nonconvex types, provided the gradients and the crossing points are computable.

2. Background Material

This section discusses background material that will be needed in the next section. It will include discussions on constraint consensus methods, binary words, and crossing points for nonlinear constraints.

We consider the system of inequality constraints of the form $f_j(x) \geq 0$, $j \in J = \{1, 2, \ldots, q\}$. For each j, f_j is a nonlinear or linear function mapping $\mathbb{R}^n$ to $\mathbb{R}$.

2.1. The Original Constraint Consensus and DBmax Constraint Consensus Methods. In this subsection, we describe the original constraint consensus and DBmax constraint consensus methods for general nonlinear constraints.

The original (basic) constraint consensus method was developed by Chinneck in [3] to find a near-feasible point for the given system. The method starts from an infeasible point x_0. The method associates each constraint $f_j(x) \geq 0$ with a *feasibility vector* d_j, defined by

$$d_j(x_0) = \frac{v(x_0)\nabla f_j(x_0)}{\left\|\nabla f_j(x_0)\right\|^2}, \tag{3}$$

where $v(x_0) = \max\{0, -f_j(x_0)\} \geq 0$ is the *constraint violation* at x_0 and $\nabla f_j(x_0)$ is the gradient of $f_j(x)$ at point x_0. We assume that $\nabla f_j(x)$ exists and that $\nabla f_j(x_0) \neq 0$ if x_0 is infeasible. If $\nabla f(x_0) = 0$ and $f_j(x_0) \geq 0$, then we define $d_j(x_0) = 0$ to resolve the ambiguity in (3). We will show in Section 3 that if f_j is linear, then $x_0 + d_j$ is the point on the boundary of $f_j(x) \geq 0$ that is closest to x_0. The length of the feasibility vector, $\|d_j(x_0)\| = v(x_0)/\|\nabla f_j(x_0)\|$, is called the *feasibility distance*, and it is an estimate of the distance to the boundary. We define x_0 to be *near-feasible* with respect to $f_j(x) \geq 0$ if $\|d_j(x_0)\| < \alpha$, where α is a preassigned feasibility tolerance. We say that x_0 is *strictly feasible* with respect to $f_j(x) \geq 0$ if $f_j(x_0) > 0$. In summary, with respect to the constraint $f_j(x) \geq 0$, we say that

(i) x_0 is near-feasible if $\|d_j(x_0)\| < \alpha$;

(ii) x_0 is feasible if $f_j(x_0) \geq 0$ or, equivalently, $d_j(x_0) = 0$;

(iii) x_0 is strictly feasible if $f_j(x_0) > 0$.

The feasibility vectors at the initial point x_0 for all the q constraints are combined to give a single *consensus vector* s_0. In the original constraint consensus method, this consensus vector is the average of all the feasibility vectors with length $\|d_j\| \geq \alpha$. However, the ith component of the consensus vector is averaged over only the subset of feasibility constraints that actually include the ith variable. The first iterate is given by $x_1 = x_0 + s_0$ and the process is repeated with $x_{k+1} = x_k + s_k$. The algorithm terminates if the number (NINF) of constraints that violates the near-feasibility condition $\|d_j\| < \alpha$ at x_k is zero. It also stops if the length of the consensus vector $\|s_k\| < \beta$, where β is a given tolerance. Note that s_k might be shorter than β even when x_0 is far from $\mathscr{R}$ if two feasibility vectors point in nearly opposite directions. Of course, the algorithm will also terminate if some maximum number of iterations is reached.

In [9], Ibrahim and Chinneck gave several variations of the original constraint consensus method. The best algorithm variant for finding near-feasible points appears to be the maximum direction-based (DBmax) constraint consensus method. Their numerical experiments show that DBmax has the best success rate for finding near-feasible points. It also

takes fewer iterations than most of the other methods. The DBmax constraint consensus method looks at the number of positive and negative values of each of the components of the feasibility vectors $d_j(x)$. For the ith component, DBmax finds the number of constraints $f_j(x) \geq 0$ with positive ith entry in their feasibility vector d_j. It also looks for the number of constraints $f_j(x) \geq 0$ with negative ith entry in their feasibility vector d_j. The sign with largest number (votes) of constraints is considered the winner.

After determining the winning sign, DBmax creates the consensus vector s_k by choosing each ith component of s_k to be the largest proposed movement in the winning direction. If a component has the same number of positive and negative votes then DBmax takes the average of the largest proposed movements in the positive and negative direction. If $(d_j)_i = 0$ for all j, then $(s_k)_i$ is set at 0. As in the case of the original consensus method, when finding the ith component of the consensus vector in DBmax, only the subset of feasibility constraints that actually includes the ith variable is considered. An example of the two methods for computing the consensus vector is as follows:

$$\begin{gathered} d_1(x_k) = \begin{pmatrix} -2 & 2 & 2 & 4 \end{pmatrix}^T \\ d_2(x_k) = \begin{pmatrix} 2 & 1 & -5 & 3 \end{pmatrix}^T \\ d_3(x_k) = \begin{pmatrix} -3 & -1 & 2 & -1 \end{pmatrix}^T \\ d_4(x_k) = \begin{pmatrix} 0 & 5 & -3 & -4 \end{pmatrix}^T \\ \hline \text{Original } s_k = \begin{pmatrix} -1 & 1.75 & -1 & 0.5 \end{pmatrix}^T \\ \text{DBmax } s_k = \begin{pmatrix} -3 & 5 & -1.5 & 0 \end{pmatrix}^T. \end{gathered} \tag{4}$$

In this example, constraint 4 does not depend on x_1; that is, $A_1^{(4)} = 0$. Thus, the first component of the original s_k is obtained by averaging −2, 2, and −3. In the DBmax calculation, the winning sign is negative for the first component; in the second component positive is the winning sign. There is a tie in the third component, so we take the average of 2 and −5. The fourth component also has no winning sign and shows a rare example where the component of the original consensus vector is larger (in absolute value) than the same component of the DBmax consensus vector.

Usually, $\|s_k\|$ is larger for the DBmax method than for the original method. For this reason, the DBmax method algorithm makes rapid progress toward the feasible region and significantly reduces the number of iterations needed to reach near feasibility as compared to the original method [9]. On the other hand, when the functions $f_j(x)$ are concave, the feasible region $\mathscr{R}$ is convex and component-averaging gradient-projection schemes (including the original constraint consensus method) have been proved to converge [3].

2.2. Binary Words and Crossing Points. For simplicity of presentation, we assume that the functions $f_j(x)$, $j \in J := \{1, 2, \ldots, q\}$ are concave over $\mathbb{R}^n$. This will insure that the feasible region is convex.

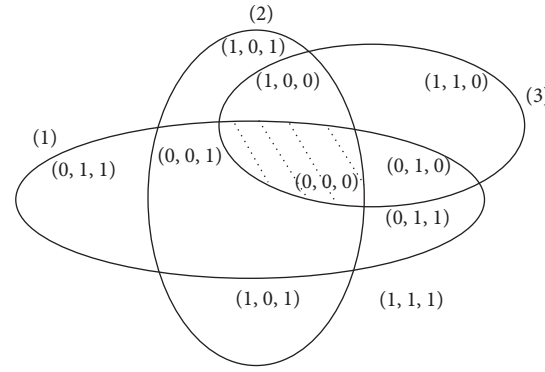

Figure 1: A partition of $\mathbb{R}^n$ formed by binary words $\delta(x)$. The feasible region, with $\delta = (0, 0, 0)$, is shaded.

For each $j \in J$, define the characteristic function $\delta_j : \mathbb{R}^n \to \{0, 1\}$ by

$$\delta_j(x) = \begin{cases} 0 & \text{if } f_j(x) \geq 0 \ \left(\text{the } j\text{th constraint is satisfied at } x\right) \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

Define the binary word function $\delta : \mathbb{R}^n \to \{0, 1\}^q$ by

$$\delta(x) = \left(\delta_1(x), \delta_2(x), \ldots, \delta_q(x)\right). \tag{6}$$

For each $x \in \mathbb{R}^n$, $\delta(x)$ is a *binary word*. If $\delta(x) = (0, 0, \ldots, 0)$, then x is a feasible point of $\mathscr{R}$. Furthermore, $\sum_{j=1}^{q} \delta_j(x) = \mathbf{1}^T \delta(x)$ is the number of violated constraints at x. Consider the equivalence relation defined by $x \sim y$ if $\delta(x) = \delta(y)$. The equivalence classes form a partition of $\mathbb{R}^n$ (or a subset of $\mathbb{R}^n$). Figure 1 shows an example with $q = 3$ constraints. The curves $f_j(x) = 0$ are labeled with (j). The interior of this boundary is the convex region where $f_j(x) > 0$. In this example there are 8 equivalence classes. In general there are at most 2^q equivalence classes.

Suppose x_k is a point in $\mathbb{R}^n$, and s_k is a direction vector (not necessarily the consensus vector). The *crossing points* of the constraint $f_j(x) \geq 0$ on the ray $\{x_k + ts_k \mid t \geq 0\}$ are $x_k + t_1 s_k$ and $x_k + t_2 s_k$, where $t_1 = \min\{t \mid f_j(x_k + ts_k) \geq 0\}$ and $t_2 = \max\{t \mid f_j(x_k + ts_k) \geq 0\}$ are positive. There are at most 2 crossing points since f is concave. For general concave constraints these optimization problems are difficult, but, in the case of linear matrix inequalities, t_1 and t_2 are solutions to $|A^{(j)}(x_k + ts_k)| = 0$; hence they are generalized eigenvalues that satisfy $|A^{(j)}(x_k) + t \sum_{i=1}^{n} A_i^{(j)}(s_k)_i| = 0$. Algorithms for finding the crossing points of an LMI can be found in [11, 12]. These algorithms depend on the sign of $f(x_k)$. When $f(x_k) < 0$, the ray has 0, 1, or 2 crossing points; see Figure 2. If $f(x_k) > 0$, the ray has at most 1 crossing point.

3. The Original Constraint Consensus and DBmax Constraint Consensus Methods for Linear Matrix Inequalities

This section discusses the tools required for the implementation of the original (basic) and DBmax constraint consensus

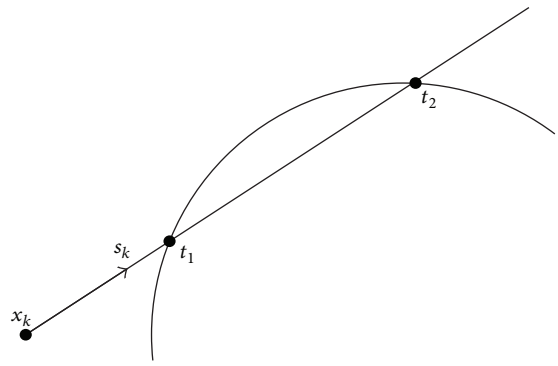

FIGURE 2: The ray $\{x_k + ts_k \mid t \geq 0\}$ and the crossing points of the constraint $f_j(x) \geq 0$ in an example where $f(x_k) < 0$.

methods described in Section 2.1 to LMI constraints. We give some theoretical results and show how to compute the gradient.

A symmetric matrix A is positive semidefinite if and only if its smallest eigenvalue, $\lambda_{\min}(A)$, is nonnegative. Thus the LMI system (1) can be written as

$$f_j(x) := \lambda_{\min}\left(A^{(j)}(x)\right) \geq 0, \quad j \in J. \tag{7}$$

For simplicity of presentation, in this section we consider a *single* LMI constraint $A(x) := A_0 + \sum_{i=1}^{n} x_i A_i \succeq 0$, which is equivalent to $f(x) = \lambda_{\min}(A(x)) \geq 0$.

Lemma 1. *The function $f(x) = \lambda_{\min}(A(x))$ is concave over $\mathbb{R}^n$.*

Proof. Theorem 3.5 in [13] states that $\lambda_{\max}(A(x))$ is convex over $\mathbb{R}^n$. Therefore $\lambda_{\min}(A(x)) = -\lambda_{\max}(-A(x))$ is concave. □

A consequence of Lemma 1 is that the feasible region for one LMI or a system of LMIs (1) is convex. Let $g(x) = \nabla_x f(x)$ be the gradient vector of $f(x) = \lambda_{\min}(A(x))$.

Lemma 2. *Suppose $x_0 \in \mathbb{R}^n$ is an infeasible point of $A(x) \succeq 0$, in which a feasible point of $A(x) \succeq 0$ exists and that $g(x)$, the gradient of $f(x)$, exists at x_0. Then, $g(x_0) \neq 0$.*

Proof. Assume that $g(x_0) = 0$. Since $f(x)$ is concave over $\mathbb{R}^n$ by Lemma 1, then $f(x) \leq f(x_0)$ for all $x \in \mathbb{R}^n$. Since x_0 is an infeasible point, $f(x_0) < 0$. Thus, $A(x) \not\succeq 0$ for all $x \in \mathbb{R}^n$. This would contradict our assumption that $A(x) \succeq 0$ is feasible. □

In the remainder of this section, we will assume that the hypotheses of Lemma 2 hold. Thus, the feasibility vector is

$$d := d(x_0) = \frac{-f(x_0)\, g(x_0)}{\left\|g(x_0)\right\|^2}. \tag{8}$$

Let $h(t) = f(x_0 + td)$, for $t \in \mathbb{R}$ be the value of $\lambda_{\min}$ along a line in $\mathbb{R}^n$, and let $L(t) = h(0) + h'(0)t$ be the linearization of h at 0.

Theorem 3. *Under the hypotheses of Lemma 2, the function $h : \mathbb{R} \to \mathbb{R}$ defined by $h(t) = f(x_0 + td)$ is concave. The linearization of h at 0, L, satisfies*

$$h(t) := f(x_0 + td) \leq L(t) = f(x_0)(1 - t), \quad \text{where } f(x_0) < 0. \tag{9}$$

In particular, $L(1) = 0$.

Proof. The restriction of a concave function on $\mathbb{R}^n$ to a line is concave, so Lemma 1 implies that h is concave. Since h is concave, $h(t) \leq L(t)$. Note that $h(0) = f(x_0)$. By Lemma 2, $\| g(x_0)\| \neq 0$. Then, by the chain rule, the slope of L is

$$h'(0) = g(x_0)^T \left(-f(x_0) \frac{g(x_0)}{\left\|g(x_0)\right\|^2}\right) = -f(x_0) > 0. \tag{10}$$

Thus, $L(t) = h(0) + h'(0)t = f(x_0) - tf(x_0)$. □

An interpretation of $L(1) = 0$ is that the normalization of d was *chosen* so that $x_0 + d$ is the linear approximation of the boundary of f. Typical graphs of h and L are shown in Figure 3.

The following corollary shows that the distance (if it exists) from x_0 along d to the boundary of $A(x) \succeq 0$ is at least $\|d\|$.

Corollary 4. *Under the hypotheses of Lemma 2, (a) $A(x_0 + td) \not\succeq 0$ for all $t < 1$. (b) Furthermore, if $A(x) \succeq 0$ is a linear constraint, then $A(x_0 + d) \succeq 0$; that is, $x_0 + d$ lies on the boundary of $A(x) \succeq 0$.*

Proof. (a) Since x_0 is infeasible, $f(x_0) < 0$. By Theorem 3, $f(x_0 + td) < 0$ for all $t < 1$, and thus $A(x_0 + td) \not\succeq 0$, for all $t < 1$. (b) If $A(x) \succeq 0$ is a linear constraint, then $\lambda_{\min}(x)$ is linear in x. Thus $h(t) = L(t)$ and $f(x_0 + d) = L(1) = 0$. Combined with part (a) this shows that $x_0 + d$ is a boundary point of $A(x) \succeq 0$. □

Remark 5. Even when $A(x) \succeq 0$ is nonlinear, it is possible that $x_0 + d$ lies on the boundary. Consider the following LMI:

$$A(x) := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + x_1 \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \succeq 0. \tag{11}$$

The eigenvalues of $A(x_1, x_2)$ are $1 \pm \sqrt{x_1^2 + x_2^2}$, so $\lambda_{\min}(A(x_1, x_2)) = 1 - \sqrt{x_1^2 + x_2^2}$ and its gradient is $g(x_1, x_2) = (-x_1/\sqrt{x_1^2 + x_2^2}, -x_2/\sqrt{x_1^2 + x_2^2})$. The feasible region is the closed unit disk. Suppose $x_0 = (x_1, x_2)$ is infeasible; that is, $x_1^2 + x_2^2 > 1$. A calculation shows that $x_0 + d = (x_1/\sqrt{x_1^2 + x_2^2}, x_2/\sqrt{x_1^2 + x_2^2})$. Thus, $x_0 + d$ is on the boundary of the feasible region, as shown in Figure 4. Theorem 3 says that $x_0 + d$ cannot be in the *interior* of the feasible region. Usually $x_0 + d$ is infeasible for nonlinear constraints. In the present example $x_0 + d$ is feasible because $h(t)$ is linear on $[0, 1]$. This motivates the following theorem.

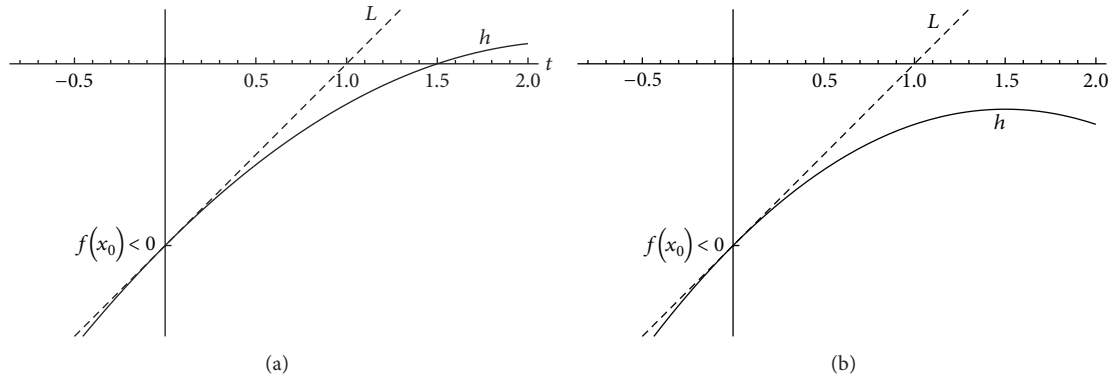

FIGURE 3: Typical graphs of $h(t) = f(x_0 + td)$ and $L(t) = f(x_0) \cdot (1 - t)$, its linearization at 0. In the left case, the ray $\{x_0 + td \in \mathbb{R}^n \mid t > 0\}$ intersects with the boundary of the feasible region at $x_0 + 1.5d$, since $h(1.5) = 0$. In the second case, this ray does not intersect with the feasible region, since h is never 0.

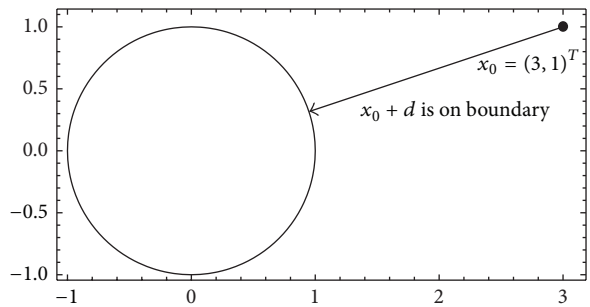

FIGURE 4: For the LMI (11), the feasible region is the disk, and $x_0 + d$ is on the boundary.

Theorem 6. *Under the hypotheses of Lemma 2, $h(t)$ is linear over $[0, 1]$ if and only if $x_0 + d$ is a boundary point of $A(x) \succeq 0$.*

Proof. Suppose $h(t)$ is linear over $[0, 1]$. Then, $h(t) = L(t)$ for $0 \leq t \leq 1$. Thus $h(1) = L(1) = 0$. Combined with Corollary 4(a), this implies that $x_0 + d$ is a boundary point of $A(x) \succeq 0$. On the other hand, assume that $x_0 + d$ is a boundary point of $A(x) \succeq 0$. Then $h(1) = 0$. Thus, the concave function h agrees with its linearization L at 0 and 1, and therefore $h(t) = L(t)$ for all $t \in [0, 1]$. □

To apply the constraint consensus methods to linear matrix inequalities, we need to be able to compute the gradient $g(x) = \nabla_x f(x)$. This is given in the following results. We assume that the matrices A_i are each of dimension $m \times m$.

Let $\{\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_m\}$ be eigenvalues of $A(x)$ with the corresponding orthonormal set of eigenvectors $\{v_1, v_2, \ldots, v_m\}$.

Lemma 7. *If $\lambda_1(A(x)) < \lambda_2(A(x))$, then the function $\lambda_{\min}(A(x))$ is differentiable at x and*

$$\frac{\partial \lambda_{\min}(A(x))}{\partial x_i} = v_1^T A_i v_1 \quad \text{for } i = 1, 2, \ldots, n. \tag{12}$$

Proof. Since $\lambda_{\min}(A(x)) = \lambda_1(A(x))$ is simple and $\lambda_{\min}(A(x)) = -\lambda_{\max}(-A(x))$, this follows from Corollary 3.10 in [13]. The derivative of $A(x)$ with respect to x_i is A_i, so

$$\frac{\partial \lambda_{\min}(A(x))}{\partial x_i} = v_1^T \frac{\partial A(x)}{\partial x_1} v_1 = v_1^T A_i v_1 \quad \text{for } i = 1, 2, \ldots, n. \tag{13}$$

□

By Lemma 7, the components of $g(x) = \nabla \lambda_{\min}(A(x))$ are given by $v_1^T A_i v_1$ $(i = 1, 2, \ldots, n)$. The above results can now be used to apply the original and DBmax constraint consensus methods in Section 2.1 to LMI constraints (1). Recall that these methods, on convergence, search for near-feasible points and nonstrictly feasible points.

4. New Constraint Consensus Algorithms

In this section, we modify and combine the original (basic) and DBmax constraint consensus methods described in [3, 9] to find strictly feasible points for our system of LMI constraints (1).

We note that strictly feasible points satisfy $A^{(j)}(x_0) \succ 0$ (i.e., $A^{(j)}(x_0)$ is positive definite) for all $j \in J$, and lie in the interior of the feasible region $\mathscr{R}$. We give four variations of the new algorithms: *Original-DBmax (OD) constraint consensus method, DBmax-DBmax (DD) constraint consensus method, Original-Original (OO) constraint consensus method and DBmax-Original (DO) constraint consensus method. Algorithm* 1 *is OD, Algorithm* 2 *is DD, Algorithm* 3 *is OO and Algorithm* 4 *is DO*. Each method has two phases: Phase 1 and Phase 2. For example, the OD method uses the original constraint consensus vector in Phase 1 and the DBmax constraint consensus vector in Phase 2.

Phase 1 is the classical constraint consensus method given in [3, 9]. The starting point in Phase 1 is a random point. Phase 1 stops when a near-feasible point is found (i.e., $\|d_j(x_k)\| \leq \alpha$ for all $j \in J$), when the consensus vector is shorter than

INPUT: An initial point x, a feasibility distance tolerance α, a movement tolerance β, maximum number of iterations N and M
Phase 1: Do the original basic constraint consensus method, Algorithm 1 in [9], using the parameters α, β and N
and starting from x to get a near-feasible point.
Let x_0 be the last iterate of Phase 1. (Remark: x_0 is the starting point of Phase 2.)
Phase 2: (uses DBmax consensus method directions given in [9])
Set $x = x_0$
Set $k = 0$
While $k < M$ and x is infeasible **do**
  Set $NINF = 0$, $n_i^+ = 0$, $n_i^- = 0$, $s_i^+ = 0$, $s_i^- = 0$ for each variable x_i
  for every constraint j **do**
    if constraint j is violated **then**
      Calculate feasibility vector d_j
      for every variable x_i in jth constraint **do**
        if $d_{ji} > 0$ **then**
          $n_i^+ \leftarrow n_i^+ + 1$
          if $d_{ji} > s_i^+$ **then**
            $s_i^+ \leftarrow d_{ji}$
        else if $d_{ji} < 0$ **then**
          $n_i^- \leftarrow n_i^- + 1$
          if $d_{ji} < s_i^-$ **then**
            $s_i^- \leftarrow d_{ji}$
  for every variable x_i: **do**
    if $n^+ = n^-$ **then**
      $s_i = (s_i^+ + s_i^-)/2$
    else if $n_i^+ > n_i^-$ **then**
      $s_i = s_i^+$
    else
      $s_i = s_i^-$
Determine the LMI crossing points $x + t_\ell s$, with $0 < t_1 < \cdots < t_{\ell^+}$, on the consensus ray $\{x + ts \in \mathbb{R}^n \mid t > 0\}$, and let j_ℓ denote the constraint of the crossing point t_ℓ.
If there are no crossing points (i.e. $\ell^+ = 0$), set $t_0 = 0$.
Set $t_{1+\ell^+} = 1 + t_{\ell^+}$.
Set $z = x + (1/2)t_1 s$
Set $\delta = \delta(x)$
Set $\widehat{\delta} = \delta$
for $\ell = 1, \ldots, \ell^+$ **do**
  Update δ by flipping δ_{j_ℓ}, the j_ℓth bit of δ
  if $\mathbf{1}^T\widehat{\delta} > \mathbf{1}^T\delta$ **then**
    replace z with $x + (1/2)(t_\ell + t_{\ell+1})s$
    Set $\widehat{\delta} = \delta$
Set $x = z$
If $\widehat{\delta} = (0, 0, \ldots, 0)$, then x is feasible
$k \leftarrow k + 1$.

Algorithm 1: Original-DBmax (OD) constraint consensus algorithm.

INPUT: an initial point x, a feasibility distance tolerance α, a movement tolerance β, maximum number of iterations N and M
Phase 1: Do the DBmax constraint consensus method, Algorithm 4 described in [9], using the parameters α, β and N and starting from x to get a near-feasible point.
Let x_0 be the last iterate of Phase 1.
Phase 2: as in the OD Constraint Consensus Algorithm

Algorithm 2: DBmax-DBmax (DD) constraint consensus algorithm.

INPUT: an initial point x, a feasibility distance tolerance α, a movement tolerance β, maximum number of iterations N and M
Phase 1: as in the OD constraint Consensus Algorithm
Phase 2: (uses original basic consensus method directions)
Set $x = x_0$
Set $k = 0$
While $k < M$ and x is infeasible **do**
  Set $n_i = 0$, $s_i = 0$ for all i
  for every constraint j **do**
    if constraint j is violated **then**
      $n_i \leftarrow n_i + 1$
      Calculate feasibility vector d_j
      for every variable x_i in jth constraint **do**
        $n_i \leftarrow n_i + 1$, $s_i \leftarrow s_i + (d_j)_i$
  for every variable x_i in jth constraint: **do**
    if $n_i \neq 0$ **then**
      $t_i = s_i/n_i$
    else
      $t_i = 0$
Determine the LMI crossing points $x + t_\ell s$, with $0 < t_1 < \cdots < t_{\ell^+}$, on the consensus ray $\{x + ts \in \mathbb{R}^n \mid t > 0\}$, and let j_ℓ denote the constraint of the crossing point t_ℓ.
If there are no crossing points (i.e. $\ell^+ = 0$), set $t_0 = 0$.
Set $t_{1+\ell^+} = 1 + t_{\ell^+}$.
Set $z = x + (1/2)t_1 s$
Set $\delta = \delta(x)$
Set $\widehat{\delta} = \delta$
for $\ell = 1, \ldots, \ell^+$ **do**
  Update δ by flipping δ_{j_ℓ}, the j_ℓth bit of δ
  if $\mathbf{1}^T\widehat{\delta} > \mathbf{1}^T\delta$ **then**
    replace z with $x + (1/2)(t_\ell + t_{\ell+1})s$
    Set $\widehat{\delta} = \delta$
Set $x = z$
If $\widehat{\delta} = (0, 0, \ldots, 0)$, then x is feasible
$k \leftarrow k + 1$.

Algorithm 3: Original-original (OO) constraint consensus algorithm.

INPUT: an initial point x, a feasibility distance tolerance α, a movement tolerance β, maximum number of iterations N and M
Phase 1: as in the DD Constraint Consensus Algorithm
Phase 2: as in the OO Constraint Consensus Algorithm

Algorithm 4: DBmax-original (DO) constraint consensus algorithm.

the movement tolerance ($\|s_k\| \leq \beta$), or when the maximum number N of iterations is reached ($k = N$).

After exiting Phase 1, the algorithm goes to Phase 2 to try to find a strictly feasible (not just near-feasible) point. The first iterate x_0 in Phase 2 is last iterate of Phase 1. Phase 2 now generates a consensus vector s_0 determined by *all* the unsatisfied constraints. In other words, the feasibility distance tolerance α is set to 0 in Phase 2. Then, all the crossing points of the constraints on the *consensus ray* $\{x_0 + ts_0 \in \mathbb{R}^n \mid t \in [0, \infty)\}$ are computed (see Figure 5), using the method described in [12].

The corresponding binary words $\delta(x)$ on each line segment in the consensus ray are determined efficiently by starting with $\delta(x_0)$ and flipping the jth bit when constraint j is crossed. This assumes that the consensus ray actually crosses a boundary curve whenever it touches the curve. The next iterate x_1 is the midpoint of the most-satisfied segment, that is, the segment with the most zeros in its binary word. Thus, $\delta(x_1) = \widehat{\delta}$ in our Algorithms 1 and 3. If the most-satisfied segment is a ray going to infinity, then the next iterate x_1 is the last crossing point plus $(1/2)s_0$. If there are no crossing points on the consensus ray, then $x_1 = x_0 + (1/2)s_0$. Algorithms 1 and 3 do this by defining t_ℓ, with $\{t \in 1, 2, \ldots, \ell^+\}$ to be the crossing points, and then adding one point $t_{1+\ell^+} = 1 + t_{\ell^+}$. If more than one segment has the maximum number of zeros in its binary word, then x_1 is in

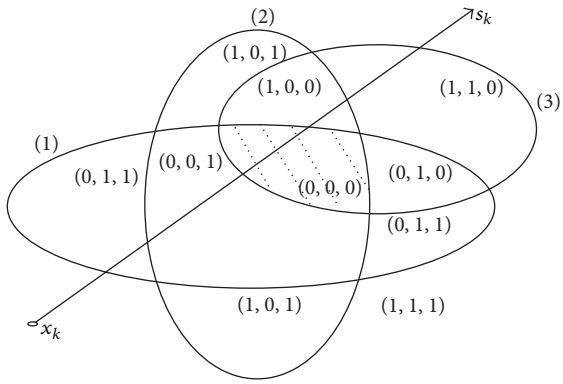

Figure 5: In Phase 2, the crossing points and binary words are computed along the consensus ray. In this example a feasible point, with binary word $\delta = (0, 0, 0)$, is found.

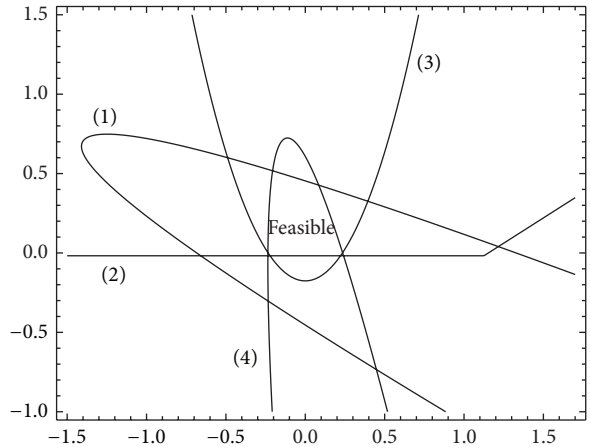

Figure 6: The boundary of each LMI constraint for Problem 1. The LMIs are given in (14). The feasible region for any LMI is determined by polynomial inequalities of degree at most m_j, which is 2 for these constraints.

the line segment closest to x_0. The process is repeated with x_1 and so on, until a feasible point of $\mathcal{R}$ is found or the maximum number M of iterations is reached.

5. Numerical Experiments I: Comparing OD, DD, OO, and DO Methods

In this section, we check the effectiveness of the four modified constraint consensus algorithms on benchmark problems, as well as many new randomly generated problems. In addition to measuring the success in finding strictly feasible points, we also compare their performances on the number of iterations and time to find a feasible point.

All numerical experiments were done using a Dell OptiPlex 980 computer with codes written in MATLAB©. The list of the test problems is given in Table 1. The columns give the number of LMI constraints q, the dimension n, and the sizes m_j of the matrices. We randomly generated problems 1 through 13, whereas the others are taken from the SDPLIBRARY [10] as listed in the last column. For each random problem, n, q, and m_j are chosen not at random. However, for each problem, LMI $A_0^{(j)} + \sum_{i=1}^{n} x_i A_i^{(j)} \succeq 0$ was generated randomly as follows: $A_0^{(j)}$ is an $m_j \times m_j$ diagonal matrix with each diagonal entry chosen from $U(0, 1)$. Each $A_i^{(j)}$ $(1 \leq i \leq n)$ is a random $m_j \times m_j$ symmetric and sparse matrix with approximately $0.8 * m_j^2$ nonzero entries generated using the Matlab command $sprandsym(m_j, 0.8)$. This ensures that, for each of the 13 random problems, the origin is an interior point. The LMIs of problem 1 are given in (14). The boundaries of the four LMI constraints are shown in Figure 6

$$
\begin{aligned}
A^{(1)}(x) &:= \begin{bmatrix} 0.4376 & 0 \\ 0 & 0.8918 \end{bmatrix} + x_1 \begin{bmatrix} 0 & 0.6556 \\ 0.6556 & 0.6325 \end{bmatrix} \\
&\quad + x_2 \begin{bmatrix} 0 & 1.3766 \\ 1.3766 & 0 \end{bmatrix} \succeq 0, \\
A^{(2)}(x) &:= \begin{bmatrix} 0.0202 & 0 \\ 0 & 0.8326 \end{bmatrix} + x_1 \begin{bmatrix} 0 & 0 \\ 0 & -0.7219 \end{bmatrix} \\
&\quad + x_2 \begin{bmatrix} 1.1393 & 0 \\ 0 & 1.1340 \end{bmatrix} \succeq 0, \\
A^{(3)}(x) &:= \begin{bmatrix} 0.1434 & 0 \\ 0 & 0.9447 \end{bmatrix} + x_1 \begin{bmatrix} 0 & 1.5723 \\ 1.5723 & 0 \end{bmatrix} \\
&\quad + x_2 \begin{bmatrix} 0.8152 & 0 \\ 0 & -0.0153 \end{bmatrix} \succeq 0, \\
A^{(4)}(x) &:= \begin{bmatrix} 0.6121 & 0 \\ 0 & 0.6169 \end{bmatrix} + x_1 \begin{bmatrix} 0 & 2.6095 \\ 2.6095 & 0 \end{bmatrix} \\
&\quad + x_2 \begin{bmatrix} 0 & 0.4044 \\ 0.4044 & -0.8521 \end{bmatrix} \succeq 0.
\end{aligned}
\tag{14}
$$

We ran each of the test problems from Table 1 with 100 random initial conditions. Each component of the initial point was chosen from a normal distribution with mean 0 and variance 10^8. We used the four algorithms with $\alpha = 0.01$ and $\beta = 0.01$. We will later discuss the effect of the choices of α and β on the performance of the algorithms. We used a maximum of $N = 500$ iterations in Phase 1 and up to $M = 10$ iterations in Phase 2. Tables 2, 3, 4, and 5 give the results of these experiments. The "Exit" column gives the percentage of the 100 trials that completed Phase 1 successfully (i.e., Phase 1 converged in fewer than N iterations). The "Feas Pt Found?" column shows the percentage of the trials that found a strictly feasible point in Phase 2. The average iterations and times for the successful trials in Phase 1 and Phase 2 are also given for this set of test problems. Note that when Phase 1 finds a strictly feasible point, then 0 iterations of Phase 2 are performed. This is the reason that the average iterations in Phase 2 are sometimes less than 1. Phase 2 rarely converged in more than 4 iterations, even when we raised the maximum to $M = 500$ iterations. Note that each of Problems 19–25 has only one constraint, and the four methods are the same. Hence, we omitted the results for these problems in Tables 3–5. The last row at the bottom of the tables gives averages for each column

Table 1: Test problems.

Problem	Dimension (n)	Numbers of constraints (q)	Size of matrices (m_j)	Source
1	2	4	[2, 2, 2, 2]	Random
2	2	5	[2, 2, 3, 3, 4]	Random
3	2	10	[2, . . . , 2]	Random
4	3	5	[3, 3, 3, 3, 3]	Random
5	2	17	[5, . . . , 5]	Random
6	20	25	[4, . . . , 4]	Random
7	20	13	[5, . . . , 5]	Random
8	20	30	[3, . . . , 3]	Random
9	20	40	[5, . . . , 5]	Random
10	10	26	[3, . . . , 3]	Random
11	15	50	[3, . . . , 3]	Random
12	15	100	[2, . . . , 2]	Random
13	7	8	[5, . . . , 5]	Random
14	21	2	[10, 5]	control1 [10]
15	66	2	[20, 10]	control2 [10]
16	136	2	[30, 15]	control3 [10]
17	174	2	[161, 174]	arch0 [10]
18	13	3	[4, 4, 6]	hinf01 [10]
19	251	1	250	gpp250-1 [10]
20	100	1	100	mcp100 [10]
21	124	1	124	mcp124-1 [10]
22	250	1	250	mcp250-1 [10]
23	10	1	30	infd2 [10]
24	101	1	100	gpp100 [10]
25	125	1	124	gpp124-1 [10]

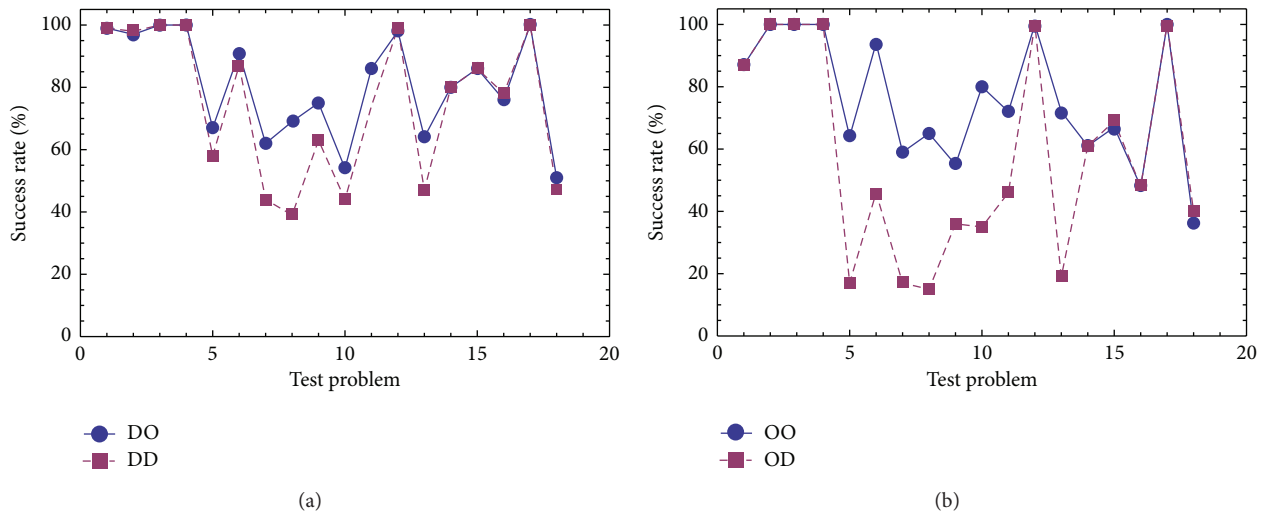

Figure 7: Comparison of the four composite methods (OO, OD, DO, and DD) for the 18 test problems that have more than one constraint. Observe that method O has a higher success rate in Phase 2.

for *all* the 25 problems, even though results for problems 19–25 are not included in Tables 3–5.

Figure 7 compares OO, OD, DO, and DD for the 18 test problems that have more than one constraint. The results show that method O usually has a higher success rate in Phase 2, regardless of which method is used in phase 1. The success rate plotted is the fraction of random initial guesses that yield a feasible point. In the left figure, method D is used in phase 1, and on the right method O is used in phase 1. The circles connected by solid lines (indicating method O in Phase 2) have a higher success rate than the squares connected by a dotted line (indicating method D in Phase 2) in almost all of

Table 2: Results of the OD method on the test problems.

Problem	Phase 1			Phase 2			Phase 1 + Phase 2
	Iter	Time (sec)	Exit	Iter	Time (sec)	Feas Pt found?	Total time (sec)
1	20.4	0.008	100%	1.0	0.002	87%	0.010
2	12.7	0.007	100%	1.0	0.002	100%	0.009
3	11.0	0.012	100%	1.0	0.003	100%	0.015
4	46.0	0.023	100%	1.4	0.002	100%	0.026
5	189.8	4.203	100%	5.2	0.188	17%	4.391
6	43.3	0.056	100%	2.8	0.008	46%	0.064
7	219.2.1	1.996	100%	5.9	0.076	17%	2.072
8	219.5	2.012	100%	6.0	0.077	15%	2.089
9	77.0	1.000	100%	5.2	0.094	36%	1.094
10	87.2	0.408	100%	3.4	0.029	35%	0.437
11	113.9	1.405	100%	4.0	0.077	46%	1.482
12	23.8	0.070	100%	1.4	0.010	100%	0.080
13	122.6	0.992	100%	5.4	0.061	19%	1.053
14	109.6	0.242	100%	1.4	0.005	61%	0.247
15	266.0	1.881	100%	1.2	0.014	69%	1.895
16	178.3	2.888	100%	1.9	0.049	48%	2.937
17	103.8	3.921	100%	1.0	0.064	100%	3.985
18	142.2	0.275	100%	1.4	0.005	40%	0.280
19	128.3	18.378	100%	1.5	0.431	78%	18.809
20	61.9	0.583	100%	1.8	0.040	100%	0.623
21	81.8	1.021	100%	4.0	0.153	66%	1.174
22	163.1	6.892	100%	3.5	0.869	64%	7.761
23	108.2	0.209	95%	1.2	0.007	73%	0.216
24	53.1	0.801	100%	1.4	0.037	65%	0.838
25	65.2	1.601	100%	1.5	0.073	72%	1.674
Average	105.9	2.035	100%	2.6	0.095	**62%**	2.130

Table 3: Results of the DD method on the test problems with more than one constraint.

Problem	Phase 1			Phase 2			Phase 1 + Phase 2
	Iter	Time (sec)	Exit	Iter	Time (sec)	Feas Pt found?	Total time (sec)
1	11.7	0.006	100%	1.1	0.002	99%	0.008
2	8.0	0.005	100%	0.9	0.002	98%	0.007
3	11.6	0.013	100%	0.9	0.003	100%	0.016
4	42.5	0.023	100%	1.1	0.002	100%	0.025
5	29.2	0.671	100%	2.7	0.099	58%	0.770
6	20.5	0.029	100%	2.0	0.006	87%	0.035
7	44.9	0.418	100%	3.5	0.045	44%	0.463
8	45.5	0.432	100%	3.5	0.049	39%	0.481
9	27.0	0.379	100%	3.4	0.072	63%	0.451
10	25.4	0.126	100%	2.0	0.018	44%	0.144
11	26.0	0.337	100%	2.2	0.043	74%	0.380
12	18.3	0.057	100%	1.1	0.008	99%	0.065
13	32.3	0.269	100%	3.4	0.039	47%	0.308
14	114.4	0.253	100%	1.4	0.005	80%	0.258
15	285.0	1.984	99%	1.6	0.018	86%	2.002
16	226.6	3.613	100%	1.2	0.032	78%	3.645
17	46.9	1.786	100%	1.0	0.073	100%	1.859
18	102.9	0.197	100%	1.9	0.006	47%	0.203
Average	71.2	1.603	100%	2.0	0.085	**74%**	1.688

Table 4: Results of the OO method on the test problems with more than one constraint.

Problem	Phase 1			Phase 2			Phase 1 + Phase 2
	Iter	Time (sec)	Exit	Iter	Time (sec)	Feas Pt found?	Total time (sec)
1	20.4	0.008	100%	1.1	0.002	87%	0.010
2	12.7	0.007	100%	1.0	0.003	100%	0.010
3	11.0	0.012	100%	1.0	0.004	100%	0.016
4	46.0	0.024	100%	1.4	0.003	100%	0.027
5	192.7	4.276	100%	2.6	0.098	64%	4.374
6	40.1	0.053	100%	2.1	0.007	94%	0.060
7	226.5	2.080	100%	2.7	0.036	59%	2.116
8	224.6	2.079	100%	2.6	0.036	65%	2.115
9	78.4	1.167	100%	3.0	0.059	55%	1.226
10	92.6	0.435	100%	2.0	0.019	80%	0.454
11	116.2	1.435	100%	2.6	0.052	72%	1.487
12	23.8	0.071	100%	1.2	0.098	100%	0.080
13	124.3	1.006	100%	2.8	0.034	72%	1.039
14	110.5	0.246	100%	1.4	0.007	61%	0.253
15	265.1	1.870	100%	1.3	0.015	67%	1.886
16	173.7	2.814	100%	1.9	0.049	48%	2.863
17	103.9	3.935	100%	1.0	0.069	100%	4.004
18	164.1	0.316	100%	1.5	0.005	36%	0.321
Average	107.5	2.053	100%	1.9	0.085	**75%**	2.138

Table 5: Results of the DO method on the test problems with more than one constraint.

Problem	Phase 1			Phase 2			Phase 1 + Phase 2
	Iter	Time (sec)	Exit	Iter	Time (sec)	Feas Pt found?	Total time (sec)
1	11.7	0.006	100%	1.2	0.003	99%	0.009
2	8.0	0.005	100%	1.0	0.002	97%	0.007
3	11.6	0.013	100%	0.9	0.003	100%	0.016
4	42.5	0.022	100%	1.1	0.002	100%	0.024
5	29.2	0.671	100%	2.4	0.090	67%	0.761
6	20.5	0.029	100%	1.9	0.006	91%	0.035
7	45.1	0.419	100%	2.8	0.036	62%	0.455
8	45.3	0.426	100%	2.6	0.035	69%	0.461
9	27.1	0.370	100%	2.6	0.050	75%	0.420
10	25.0	0.123	100%	2.1	0.017	54%	0.140
11	26.0	0.340	100%	2.0	0.039	86%	0.379
12	18.3	0.057	100%	1.2	0.009	98%	0.0.066
13	32.1	0.267	100%	2.7	0.031	64%	0.298
14	115.3	0.255	100%	1.4	0.005	80%	0.260
15	285.0	1.983	99%	1.6	0.018	86%	2.001
16	225.5	3.598	100%	1.3	0.033	76%	3.631
17	46.9	1.828	100%	1.0	0.066	100%	1.894
18	98.7	0.191	100%	1.6	0.005	51%	0.196
Average	71.0	1.604	100%	1.9	0.082	**79%**	1.686

the cases. Comparison of the two methods that use O in Phase 2 is given in Figure 8. In the figure, the circles use method O in phase 1, and the squares use method D in Phase 1. The success rate is slightly better with DO, and the total time is much better with DO. Since Figure 7 shows that O is clearly better in Phase 2, we conclude that DO is the best of the four methods (OO, OD, DO, and DD) we tested. So, we see from the results in Tables 2–5 and Figures 7 and 8 that DO is the most successful and OD is the least successful. On the whole, the DBmax method works better in Phase 1, and the

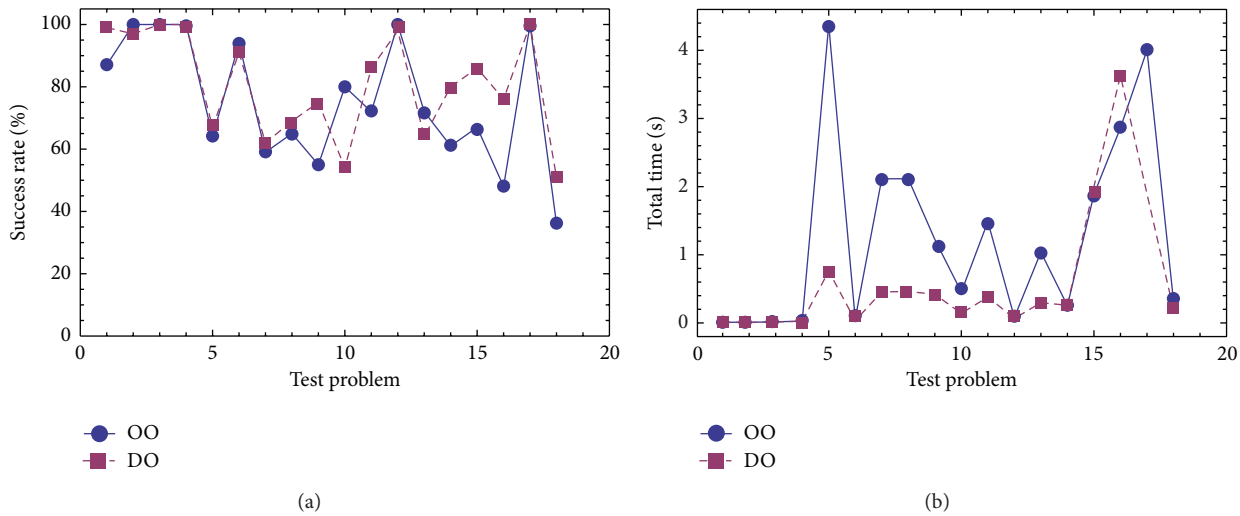

Figure 8: Comparison of the two methods that use O in Phase 2, for the same 18 test problems considered in Figure 7. We conclude that DO is the best method.

Table 6: Success rate percentages and average times for the 500 random problems.

Algorithm	Success rate	Average time (sec)
OD	31.6%	1.336
DD	46.6%	0.527
OO	50.8%	1.343
DO	**54.2%**	**0.502**

original method is better in Phase 2. Note that DO finds a strictly feasible point with average success rate of 79% for the 25 problems and 75.7% for the 12 benchmark problems.

However, the number of problems used is too small to make a definitive comparison. To compare these methods better, we generated and used a set of **500** random problems. For each problem, n is an integer uniform in the interval $[2, 30]$ and q is an integer uniform in the interval $[1, 40]$. For $1 \le j \le q$, m_j is an integer uniform in the interval $[1, 5]$. In all of the 500 problems, each LMI $A_0^{(j)} + \sum_{i=1}^{n} x_i A_i^{(j)} \succeq 0$ was generated as follows: $A_0^{(j)}$ is an $m_j \times m_j$ diagonal matrix with each diagonal entry chosen from $U(0, 1)$. Each $A_i^{(j)}$ $(1 \le i \le n)$ is a random $m_j \times m_j$ symmetric and sparse matrix with approximately $0.8 * m_j^2$ nonzero entries generated using the Matlab command $sprandsym(m_j, 0.8)$. This ensures that each of our test problems is random and that the origin is an interior point for each problem.

For each of the 500 random problems, one starting point was chosen at random and used for each of our four algorithms. Each component of the initial point was chosen from a normal distribution with mean 0 and variance 10^8. The results are given in Table 6. They confirm that the DO algorithm is the most successful in finding strictly feasible points. Another interesting result of this experiment, not shown in Table 6, is that the DBmax method in Phase 1 found a strictly feasible point in 5 out of the 500 problems, whereas the original method in Phase 1 never found a strictly feasible point in any of these 500 problems.

The results show that DO is the best method (in percentage of successes), followed by OO, followed by DD, and followed by OD. In other words, D is better in Phase 1 and O is better in Phase 2. This suggests that the original consensus directions are better than DBmax search directions for iterates close to the feasible region while the reverse is the case for iterates far away from the feasible region. The fastest is DO, followed by DD, followed by OD, and followed by OO.

To obtain some geometrical understanding of how Phase 2 can fail, we include detailed figures of problem 1. This problem also allows a visual comparison of the original and DBmax methods. Figure 9 shows how the consensus vector is computed at the initial point x_0 and shows the first few steps of Phase 1. Constraint 2 is satisfied at this point. The original consensus vector is the average of the three feasibility vectors d_1, d_3, and d_4, whereas the DBmax consensus vector has the x_1 coordinate of d_4 and the x_2 coordinate of d_1. The middle figure shows the first 8 iterations of the original constraint consensus method and the right figure shows the first 4 iterations of the DBmax method. Note how the original method converges more slowly, whereas the DBmax method follows a more erratic path. The erratic path of the DBmax method suggests that the original method gives a consensus ray (used in Phase 2) that is more likely to intersect the feasible region.

It is notable that Phase 2 does *not* always converge in one iteration when Phase 1 stops at a near-feasible point. For problem 1 this is seen in Figure 10, which shows the end of Phase 1 and Phase 2 for the OO algorithm with $\alpha = \beta = 0.01$. In this figure, Phase 1 points are dots and Phase 2 points are circles. The left figure starts with x_8 of Phase 1, and the points x_1 and x_2 of Phase 2 are visible. The detailed figure on the right shows the end of Phase 1 and the initial point of Phase 2. Note that x_{14} is within distance $\alpha = 0.01$ of the constraint 4 boundary, so the consensus vector at that point is

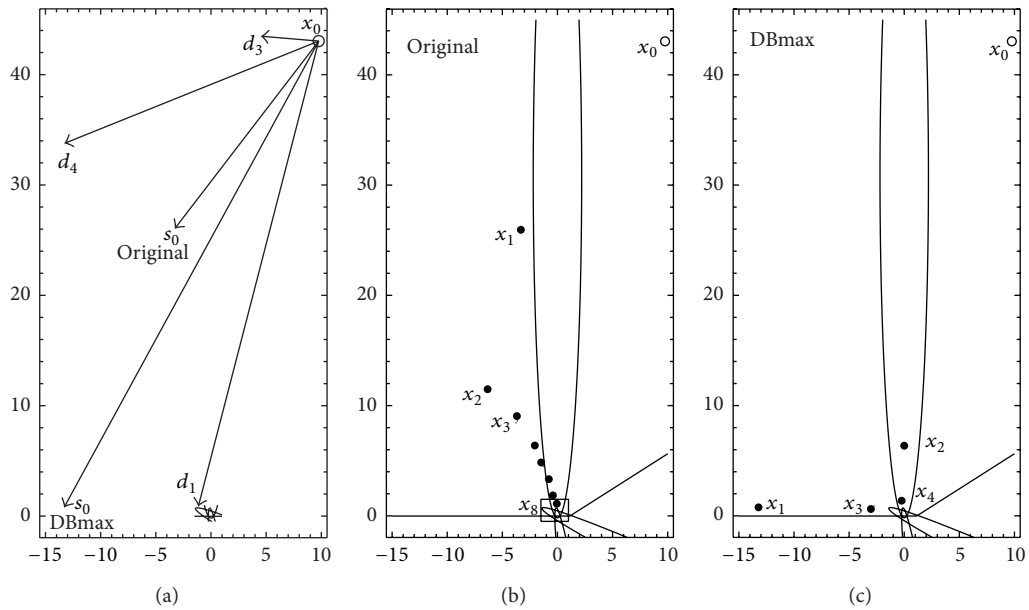

FIGURE 9: (a) shows the original consensus vector and the DBmax consensus vector for problem 1 at the point $x_0 = (9 : 7627; 43 : 0379)$. Figures (b) and (c) show phase 1 for the two methods.

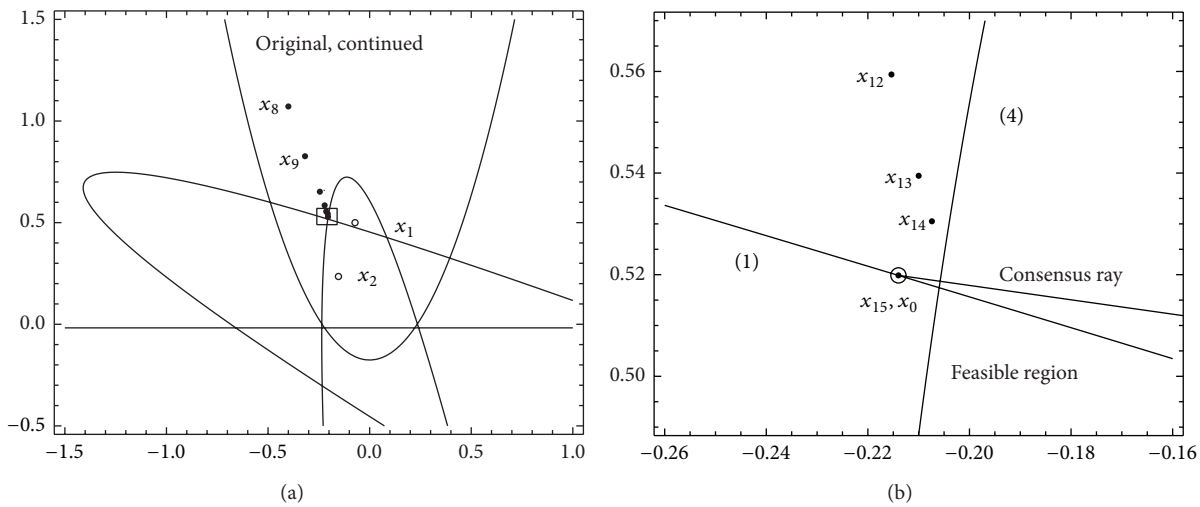

FIGURE 10: The OO algorithm for problem 1, with $\alpha = \beta = 0.01$, continued from Figure 9(b).

$s_{14} = d_1(x_{14})$, the feasibility vector for constraint 1. As a result, x_{15} is almost on the boundary of constraint 1, and Phase 1 ends with all constraints satisfied (2 and 3) or approximately satisfied (1 and 4). The final point (x_{15}) of Phase 1 is the initial point (x_0) of Phase 2, as indicated by the circle.

In Figure 10, the consensus vector calculated at x_0 of Phase 2 is the average of $d_1(x_0)$ and $d_4(x_0)$, but it is dominated by $d_4(x_0)$, and the consensus ray does not intersect the feasible region. For this problem, Phase 2 finds a feasible point in two steps as seen in the figure on the left, but for other problems one step of Phase 2 can move the point very far from the feasible region. In practice we find that Phase 2 either converges in 3 or fewer steps or it does not converge at all. In this example, the consensus rays from x_8 to x_{13} all intersect the feasible region. If β were 0.02, then Phase 1 would have stopped at x_{12}, and Phase 2 would have found a strictly feasible point in one iteration. The figure shows how the consensus ray can miss the feasible region because the initial point of Phase 2 is *too* close to the feasible region. The consensus ray based on the DBmax consensus vector misses the feasible region more often than the consensus ray based on the original consensus vector, which is why the original method is preferred in Phase 2.

The example shown in Figures 9 and 10 suggests that $\alpha < \beta$ is preferred. To further investigate the interplay between the parameters α and β, we ran the DO algorithm on problems 14 and 18 with 100 randomly chosen initial conditions, for several choices of α and β. The results are shown in Tables

Table 7: Success rate percentages for the DO method in problem 14 as α and β vary.

$\alpha \setminus \beta$	0.00001	0.0001	0.001	0.01	0.1
0.00001	65	74	**82**	80	78
0.0001	71	72	81	80	78
0.001	78	78	79	80	78
0.01	78	78	78	78	78
0.1	74	74	74	74	75

Table 8: Success rate percentages for the DO method in problem 18 as α and β vary.

$\alpha \setminus \beta$	0.00001	0.0001	0.001	0.01	0.1
0.00001	55	46	45	55	48
0.0001	55	51	54	53	46
0.001	51	38	52	**62**	49
0.01	44	45	42	60	55
0.1	41	52	43	37	55

7 and 8. The case $\alpha = 0.00001$ and $\beta = 0.001$ works best for problem 14. But, for problem 18, the best choice is $\alpha = 0.001$ and $\beta = 0.01$. These show that the optimal values of α and β are problem dependent, but $\alpha < \beta$ in our two example. In Chinneck and Ibrahim's papers [3, 9], where the goal is to find a near-feasible point in Phase 1, β is usually chosen to be much smaller than α.

6. Numerical Experiments II: Comparing DO with Another Method

In this section, we compare our DO method with the method of alternating projections (MAP) given as Algorithm 3.1 in the paper [6]. The MAP method works well in finding feasible points for LMIs and has been found to perform even better than SEDUMI [14] on certain types of LMI constraints. It has the added advantage that it allows the user to input a starting point, which enables us to compare it with DO from the same starting points.

A brief description of the MAP method is as follows. Recall that our problem is to find an interior point of the system

$$A_0^{(j)} + \sum_{i=1}^{n} x_i A_i^{(j)} \succeq 0, \quad (j = 1, \ldots, q). \tag{15}$$

The system is equivalent to the single LMI

$$F_0 + \sum_{i=1}^{n} x_i F_i \succeq 0, \tag{16}$$

where $F_0 = \mathbf{diag}(A_0^{(1)}, \ldots, A_0^{(q)})$ and $F_i = \mathbf{diag}(A_i^{(1)}, \ldots, A_i^{(q)})$ are block diagonal matrices. The matrix F_i is of size $m = \sum_{j=1}^{q} m_j$.

Consider the affine space $\mathcal{A}$ associated with the single LMI constraint $\mathcal{A} = \{F_0 + \sum_{i=1}^{n} x_i F_i \mid x \in \mathbb{R}^n\}$ and the set $\mathcal{S}_m^+$, where $\mathcal{S}_m^+$ is the set of all $m \times m$ positive semidefinite matrices. Algorithm 3.1 (MAP) in [6] seeks a point in the interior of the intersection of the two closed convex sets $\mathcal{A}$ and $\mathcal{S}_m^+$. The algorithm generates a sequence of points that converges into the interior by alternating projections onto the sets. In our implementation of MAP, we exploited the block-diagonal structures of $F_i's$ to reduce computation time. The starting point in MAP can be any $m \times m$ symmetric matrix. To allow comparison of MAP with DO, we picked starting points for MAP by choosing random $x_0 \in \mathbb{R}^n$ (as in Section 5) and using $S_1 = F_0 + \sum_{i=1}^{n} (x_0)_i F_i$ as the starting symmetric matrix.

In their paper, the authors also used a conification procedure that turns the problem into that of finding a point in the interior of the two closed convex cones $\overline{\mathcal{A}} = \{(rF_0 + \sum_{i=1}^{n} x_i F_i, r) \mid x \in \mathbb{R}^n, r \in \mathbb{R}^+\}$ and $\mathcal{S}_m^+ \times \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of positive real numbers. We will refer to Algorithm 3.1 applied to cones $\overline{\mathcal{A}}$ and $\mathcal{S}_m^+ \times \mathbb{R}^+$ as the MAPCON method. We choose starting points for MAPCON by setting $r = 1$ and choosing random $x_0 \in \mathbb{R}^n$ (as in Section 5) to give the starting symmetric matrix $S_1 = F_0 + \sum_{i=1}^{n} (x_0)_i F_i$. It has been proved in [6] that MAPCON converges after a finite number of iterations, while MAP only guarantees asymptotic convergence.

We compared DO and MAP on the 25 problems given in Section 5 and for each method ran 10 trials for each problem and the averages computed. We also used the same parameters for DO as in Section 5, that is, $\alpha = 0.01$ and $\beta = 0.01$, and used a maximum of $N = 500$ iterations in Phase 1 and up to $M = 10$ iterations in Phase 2. To avoid too long computation time, DO is stopped after reaching a maximum of 500 seconds. For MAP and MAPCON, a maximum of 510 iterations and a maximum of 500 seconds were used. For convergence, both methods are stopped when two consecutive iterates are within a tolerance distance of 0.01.

We used the same starting points for DO, MAP, and MAPCON and generated the points as in Section 5. The results for the 25 problems are given in Table 9. DO found a feasible point in each of the 25 problems, MAP in 16 problems, and MAPCON in 16 problems. For better comparison, the averages for the iterations and times were calculated over the 12 problems, where DO, MAP, and MAPCON found a strictly feasible point. In the table, we see that MAP and MAPCON were unable to find a strictly point after reaching the iteration or time limits on some of the problems. The asterisk "$*$" in Table 9 indicates the entry is irrelevant and was not recorded since either the iteration or time limit has been reached.

We also compared the methods on a set **A** of **500** random problems, which were generated in the same manner as the 500 random problems of Section 5. But, here, we decided to use smaller size problems to reduce the total time taken to complete the experiments as MAP and MAPCON were taking too much on larger values of n and q. For each problem, n is an integer uniform in the interval $[2, 10]$ and q is an integer uniform in the interval $[2, 100]$. For $1 \leq j \leq q$, m_j is an integer uniform in the interval $[1, 5]$. The results are given in Table 10. DO found a feasible point in 443 problems, MAP in 240 problems, and MAPCON in 238 and both methods were successful in 143 same problems. The averages for the

Table 9: Results of the DO, MAP, and MAPCON methods on the 25 test problems.

Problem	DO			MAP			MAPCON		
	Feas Pt found?	Iter	Time (sec)	Feas Pt found?	Iter	Time (sec)	Feas Pt Found?	Iter	Time (sec)
1	100%	13.0	0.044	100%	35.0	0.030	40%	7.0	0.016
2	100%	8.5	0.045	100%	76.8	0.068	70%	12.7	0.022
3	100%	12.6	0.073	100%	37.4	0.069	100%	45.8	0.108
4	100%	43.5	0.096	100%	91.6	0.108	20%	11.0	0.023
5	100%	15.7	0.110	100%	35.2	0.108	100%	79.6	0.308
6	40%	33.8	0.815	0%	*	>510	0%	>510	*
7	70%	46.1	0.621	30%	71.7	3.832	50%	38.2	2.417
8	60%	47.8	1.421	100%	80.6	8.934	0%	>510	*
9	60%	30.0	1.214	0%	>510	*	60%	149.0	25.418
10	50%	26.8	0.439	0%	>510	*	100%	67.0	2.173
11	70%	27.3	1.304	0%	>510	*	0%	>510	*
12	70%	33.1	2.212	0%	>510	*	10%	113.0	24.593
13	100%	23.6	0.138	100%	45.8	0.264	90%	46.1	0.322
14	80%	108.8	0.803	0%	>510	*	0%	>510	*
15	100%	261.3	6.676	0%	>510	*	0%	>510	*
16	80%	207.6	18.719	0%	*	>500	0%	*	>500
17	100%	47.2	13.128	0%	*	>500	0%	*	>500
18	20%	24.0	0.168	100%	187.6	2.428	0%	>510	*
19	60%	128.7	239.697	70%	7.0	168.965	40%	10.0	192.640
20	100%	65.4	3.059	100%	3.0	1.471	100%	3.0	1.538
21	70%	84.4	6.064	100%	3.0	2.644	100%	3.0	2.692
22	70%	163.1	84.398	100%	3.0	38.916	100%	3.0	38.346
23	80%	91.5	0.517	50%	370.6	4.250	0%	>510	*
24	90%	53.9	6.693	100%	25.2	13.662	70%	114.7	63.193
25	70%	68.7	16.433	100%	31.4	39.071	70%	181.9	201.696
Average	**77%**	56.8	29.738	**58%**	32.9	22.1	**45%**	43.1	41.742

Table 10: Results of DO, MAP, and MAPCON methods for the set **A** of **500** random problems.

Algorithm	Success rate	Average iterations	Average time (sec)
DO	89%	23.9	0.183
MAP	48%	55.0	0.500
MAPCON	48%	61.5	1.200

Table 11: Results of DO, MAP, and MAPCON methods for the set **B** of 500 random problems.

Algorithm	Success rate	Average iterations	Average time (sec)
DO	93%	16.6	0.122
MAP	41%	35.7	0.238
MAPCON	49%	205.2	1.876

iterations and times were calculated over the 143 problems where DO, MAP, and MAPCON found a strictly feasible point.

To see the effect of the size of n relative to q on our comparison of the three methods, we generated another set **B** of 500 randomly generated problems. Set **B** has fewer variables (smaller n) and more constraints (larger q) than set **A**. The results are given in Table 11. For each problem, n is an integer uniform in the interval $[2, 5]$ and q is an integer uniform in the interval $[50, 100]$. For $1 \leq j \leq q$, m_j is an integer uniform in the interval $[1, 5]$. DO found a feasible point in 467 problems, MAP in 206 problems, and MAPCON in 244 and both methods were successful in 125 problems. The averages for the iterations and times were calculated over the 125 problems where DO, MAP, and MAPCON found a strictly feasible point.

The results in Tables 9, 10, and 11 show that DO has a higher success rate than MAP and MAPCON. The DO method also took less number of iterations and time to find a strictly feasible point. It indicates that DO outperforms both MAP and MAPCON as a LMI feasibility method on our test problems. The results in Tables 10 and 11 also suggest that the performance of DO over MAP and MAPCON gets better when the number of variables is small relative to the number of LMI constraints. An unexpected consequence from our

results is that MAPCON may not be a better method than MAP in practice.

7. Conclusion

We apply, modify, and combine the constraint consensus methods of Chinneck and Ibrahim [3, 9] to find strictly feasible points of linear matrix inequality constraints. Phase 1 of our algorithm is the standard constraint consensus method, using either the original (O) or DBmax (D) consensus vector. At each step in Phase 2, we search the ray defined by the original or DBmax consensus vector for crossing points of the LMI constraints and move to the line segment on that ray where the most constraints are satisfied. We tested the four resulting algorithms, namely, *Original-DBmax (OD) constraint consensus method*, *DBmax-DBmax (DD) constraint consensus method*, *Original-Original (OO) constraint consensus method*, and *DBmax-Original (DO) constraint consensus method*. Our numerical experiments indicate that DO is the best method. It works well for the benchmark problems, finding a strictly feasible point with a success rate of 75.7% on average.

Further modifications could improve our algorithms. For example, if Phase 2 does not converge in 5 iterations it will most likely not converge. In this case, the algorithm could give up and restart Phase 1 with a new, randomly chosen starting point. Heuristics for a good choice of α and β could improve success. Another possible modification is an algorithm that gradually makes a transition from Phase 1 to Phase 2. We compared DO with the method of alternating projections and found DO to outperform it on average, especially on problems with large number of constraints relative to the number of variables. We plan in the future to study DO performance as the sizes of the variables vary and to compare it with other methods such as SEDUMI and SDPT3.

Our four algorithms can be used with any differentiable constraints, provided the crossing points of each constraint boundary with a ray can be computed. The algorithms do not require the feasible region to be convex. We have implemented the algorithms for linear matrix inequalities, where the computation of crossing points is relatively straight forward.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] M. J. Todd, "Semidefinite optimization," *Acta Numerica*, vol. 10, pp. 515–560, 2001.

[2] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM Review*, vol. 38, no. 1, pp. 49–95, 1996.

[3] 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.

[4] R. B. Kearfott and J. Dian, "An iterative method for finding approximate feasible points," Tech. Rep., Department of Mathematics, University of Southern Louisiana, Lafayette, La, USA, 2000.

[5] P. Gahinet and A. Nemirovski, "The projective method for solving linear matrix inequalities," *Mathematical Programming*, vol. 77, pp. 163–190, 1997.

[6] M. A. Rami, U. Helmke, and J. B. Moore, "A finite steps algorithm for solving convex feasibility problems," *Journal of Global Optimization*, vol. 38, no. 1, pp. 143–160, 2007.

[7] D. Baang, "Improved ellipsoid algorithm for LMI feasibility problems," *International Journal of Control, Automation and Systems*, vol. 7, no. 6, pp. 1015–1019, 2009.

[8] Y. Censor and S. A. Zenios, *Parallel Optimization: Theory, Algorithms, and Applications*, Oxford University Press, New York, NY, USA, 1997.

[9] 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.

[10] B. Borchers, "SDPLIB 1.2, library of semidefinite programming test problems," *Optimization Methods and Software*, vol. 11, no. 1–4, pp. 683–690, 1999.

[11] S. Jibrin, "Generating uniform points on the boundary of bounded spectrahedron with applications to rank constrained linear matrix inequality problem," *International Journal of Applied Mathematics & Engineering Sciences*, vol. 1, no. 1, pp. 27–38, 2007.

[12] 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.

[13] M. L. Overton and R. S. Womersley, "Optimality conditions and duality theory for minimizing sums of the largest eigenvalues of symmetric matrices," *Mathematical Programming*, vol. 62, no. 2, pp. 321–357, 1993.

[14] J. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," *Optimization Methods and Software*, vol. 11-12, pp. 625–653, 1999.