A Random Algorithm for Semidefinite Programming Problems

Jianjun Yuan and Andrew Lamperski

Abstract— **We introduce a first-order method for solving semidefinite programming problems. This method has low computational complexity per iteration and is easy to implement. In each iteration, it alternates in two steps: gradient-descent to optimize the objective function, and random projection to reduce the infeasibility of the constraints. Due to its low computational complexity per iteration, it can be scaled to large problems. We also prove the algorithm's convergence and demonstrate its performance in numerical examples.**

I. Introduction

Semidefinite programming (SDP) is common in both theory and practical applications. Many convex problems can be converted to SDP problems, and many non-convex problems can be relaxed to SDP problems. For example, controller analyis and design problems including H_∞ performance, passivity analysis, and multiobjective synthesis can be transformed into SDPs [1]. Many NP-hard combinatorial optimization problems, like Boolean least-squares and Max-Cut, can be approximated using SDP problems by Lagrangian relaxation [2].

There are many algorithms for solving SDP problems. The most famous class of algorithms uses the interior-point method [3]. The basic idea behind it is to impose some 'barrier' function to replace the constraints, and then it solve a sequence of unconstrainted problems by Newton's method. Algorithms based on Newton iteration are called "second-order" methods, because they utilize second derivatives. The interior-point method is efficient for small SDP problems. However, the Newton steps become prohibitively slow as the problem dimension increases.

More recently, first-order methods have become popular due to their scalability to larger problems. First-order methods only rely on gradient information. For example, the algorithm Splitting Conic Solver (SCS) [4], is a first-order algorithm for conic programming based on the alternating direction method of multipliers (ADMM) [5], [6]. In comparison to second-order methods, the iterations of first order methods are significantly faster, but more iterations are required for convergence.

In this paper, we develop a randomized first-order algorithm for solving semidefinite programming problems. In each iteration, our proposed algorithm alternates in two steps: objective function optimization by gradient-descent, and random projection onto constraints to reduce infeasibility. Random feasbility updates have long been employed in optimization problems. For example, [7] used this idea to solve the convex feasibility problems while an inexact projection method was proposed in [8].

The main idea of our proposed algorithm comes from the paper [9]. That work describes the general randomized algorithm for solving convex optimization problems. In this work, we view the linear matrix inequality arising in a semidefinite program as an infinite collection of linear constraints. While [9] gives convergence theory for the case of infinite constraints, it does not explain how to create a practical algorithm in this case. The key challenge with infinite constraints is selecting the constraints for projection. This paper introduces a selection mechanism based on approximate eigenvalue computations [10].

This paper is organized as follows. The problem formulation and the motivation are discussed in Section II. Section III presents the algorithm. The convergence proof is given in Section IV. Section V describes numerical experiments. Finally, Section VI contains some concluding remarks.

A. Notation

If x is a random variable, its expected value is denoted by $\mathbb{E}[x]$. $P(A)$ is the probability of event A. For matrix M, its transpose is denoted by M^T. $\mathbf{S}^k$ is the set of $k \times k$ symmetric matrices. We denote $M \preceq 0$ to denote that the matrix M is symmetric and negative semidefinite. The 2-norm of a vector x is denoted by $\|x\|_2$. The distance between a vector x and a set $\mathbf{X}$ is denoted by $dist(x, \mathbf{X})$, and is defined mathematically by $dist(x, \mathbf{X}) = \min_{y \in \mathbf{X}} \|x - y\|_2$. For a scalar g, g^+ is equal to $\max\{g, 0\}$.

II. Problem Formulation and Motivation

We focus on a general semidefinite programming problem ($\mathbf{SDP}$) of the form:

$$\min_{x \in \mathbf{X}_0} \quad c^T x \tag{1a}$$

$$s.t. \quad M(x) = M_0 + x_1 M_1 + ... + x_n M_n \preceq 0 \tag{1b}$$

$$Ax = b \tag{1c}$$

$$Fx \leq g \tag{1d}$$

where the inequality $\preceq$ represents a linear matrix inequality (LMI), M_0, M_1, …,$M_n \in \mathbf{S}^k$, $A \in \mathbf{R}^{p \times n}$, $F \in \mathbf{R}^{q \times n}$, and $x = [x_1, x_2, \dots, x_n]^T \in \mathbf{R}^n$. Here $\mathbf{X}_0$ is a convex set. In practice, $\mathbf{X}_0$ will typically be $\mathbf{R}^n$, but for the formal convergence proof we will assume that $\mathbf{X}_0$ is a compact set.

Semidefinite programming algorithms that scale to large problems are desirable. As discussed in the introduction, semidefinite programming is used in applications such as control system design [11], and combinatorial optimization

The authors are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA, 55455
yuanx270@umn.edu, alampers@umn.edu

[12], [13]. Currently, semidefinite programming has several polynomial-time algorithms [4], [14]–[16]. However, the computational complexity per iteration is high compared to other convex problems such as linear programming and quadratic programming.

For the semidefinite program from (1), the iteration complexity of an interior point method will be of order $\max\{nk^3, n^2k^2\}$ [17]. For first-order methods such as SCS, the iteration complexity is of order $\max\{nk^2, k^3\}$, where nk^2 term comes from forming the matrix $M(x)$, while the k^3 term comes from computing an eigenvalue decomposition of $M(x)$, [4], [16]. The randomized algorithm proposed in this paper has iteration complexity of $O(nk^2)$, where the cost of nk^2 comes from constructing $M(x)$. We avoid the $O(k^3)$ complexity by utilizing a randomized eigenvalue decomposition method with iteration complexity of $O(k^2)$.

III. A RANDOM ALGORITHM

Our proposed random algorithm is an iterative method with gradient-based updates for the objective and random projections for the constraints.

A. *Reformulating the* **SDP**

In this subsection, we show how to reformulate the original **SDP**, (1), to a problem with only linear equalities and inequalities. Applying the definition of a linear matrix inequality, we get the following equivalent form:

$$\min_{x\in\mathbf{X}_0} \quad c^Tx \tag{2a}$$
$$s.t. \quad v^TM(x)v \le 0, \text{ for all } v \text{ such that } \|v\|_2 = 1 \tag{2b}$$
$$Ax = b \tag{2c}$$
$$Fx \le g. \tag{2d}$$

Note that for each vector v, the quadratic form can be expressed as $v^TM(x)v = (p^v)^Tx - q^v$, where $p_i^v = v_i^TM_iv_i$ and $q = -v^TM_0v$. Thus, we have reduced the problem of checking a single linear matrix inequality, $M(x) \preceq 0$, to checking an infinite number of linear inequalities. We will see that random projection of the constraints from (2) leads to a convergent algorithm. However, naïvely randomizing over v leads to slow convergence, as most values of v will not correspond to active constraints.

In order to get a faster algorithm, we recognize that $M(x) \preceq 0$ is equivalent to requiring that all of eigenvalues of $M(x)$ are non-positive. Thus, it suffices to enforce the constraint (2b) for only the eigenvectors of $M(x)$. Denoting the eigenvectors by $v_1, \dots, v_k$, we see that original **SDP** (1) is equivalent to

$$\min_{x\in\mathbf{X}_0} \quad c^Tx \tag{3a}$$
$$s.t. \quad v_i^TM(x)v_i \le 0, i = 1, 2, \dots, k \tag{3b}$$
$$Ax = b \tag{3c}$$
$$Fx \le g. \tag{3d}$$

B. *Approximate Eigenvectors*

Computing the exact eigenvectors of $M(x)$ is computationally expensive. This subsection describes a variant of Oja's algorithm [10] to compute approximate eigenvectors of a constant matrix $M \in \mathbf{S}^k$. While the original theory from [10] requires distinct eigenvalues, we will sketch how that assumption can be relaxed. Later, we will incorporate this procedure into a general semidefinite programming algorithm. The procedure will converge to a collection of eigenvectors, v_i such with eigenvalues are ordered as:

$$\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_k. \tag{4}$$

Initialization: Initialize a random guess for the eigenvectors:

$$V^0 = \begin{bmatrix} v_1^0 & v_2^0 & \cdots & v_k^0 \end{bmatrix}$$

Here the vectors v_i^0 are distributed independently and uniformly over the unit sphere so that $\|v_i^0\|_2 = 1$.

Vector Selection: At step $t \ge 1$, we randomly select one of the vectors v_i^{t-1} to update. Convergence is guaranteed as long as all of the vectors have a positive probability of being selected. For the application of semidefinite programming, we are interested in testing if $M \preceq 0$, and thus it is important to get an accurate estimate of the highest eigenvalues. In order to emphasize the higher eigenvalues, we compute the approximate eigenvalues as:

$$\lambda_i^{t-1} = (v_i^{t-1})^TMv_i^{t-1}. \tag{5}$$

Then we select the vector v_i^{t-1} with probability proportional to $e^{\lambda_i^{t-1}/\tau}$, where $\tau > 0$ is a temperature parameter.

Vector Update: The selected vector v_i^{t-1} is updated as follows:

$$\text{Set} \quad U = \begin{cases} \begin{bmatrix} v_1^{t-1} & v_2^{t-1} & \cdots & v_{i-1}^{t-1} \end{bmatrix} & \text{if } i > 1 \\ 0 & \text{otherwise} \end{cases} \tag{6a}$$
$$\text{Set} \quad v = v_i^{t-1} - U(U^Tv_i^{t-1}) \tag{6b}$$
$$\text{Set} \quad v = v + \beta_tMv \tag{6c}$$
$$\text{Set} \quad v_i^t = \frac{v}{\|v\|_2} \tag{6d}$$

Here β_t is a step size parameter which satisfies $\sum_{t=1}^{\infty}\beta_t = \infty$ and $\sum_{t=1}^{\infty}\beta_t^2 < \infty$.

The other vectors are updated as $v_j^{t-1} = v_j^t$. The approximate eigenvalues are updated by using (5) for the selected index, i, and setting $\lambda_j^t = \lambda_j^{t-1}$ for all of the other indices.

Proposition 1: With probability 1*, the vectors v_i^t converge to an orthogonal collection of eigenvectors of M as $t \to \infty$ with corresponding eigenvalues ordered as in* (4)*. Furthermore, each iteration has complexity $O(k^2)$.*

Proof Sketch: Based on standard results of stochastic approximation, [18], the step-size choice ensures that v_1^t converges to $\lim_{\sigma\to\infty}\frac{e^{M\sigma}v_1^0}{\|e^{M\sigma}v_1^0\|_2}$. Recall that v_1^0 was uniformly distributed over the unit sphere. Thus, with probability 1 the projection of v_1^0 onto the the eigenspace of the highest eigenvalue, λ_1 is non-zero. Thus, with probability 1, v_1^t converges to a vector in this eigenspace.

Now assume inductively that for $i \geq 2$, $v_1^t, \dots, v_{i-1}^t$ converge to eigenvectors $v_1, \dots, v_{i-1}$. The projection operation from (6b) ensures that v_i^t will converge to $\lim_{\sigma\to\infty} \frac{e^{M\sigma}v}{\|e^{M\sigma}v\|_2}$, for some v which is orthogonal to $v_1, \dots, v_{i-1}$. The random initialization ensures that with probability 1, v_i^t has a non-zero projection onto the eigenspace of λ_i. It follows that v_i^t converges to an eigenvector v_i corresponding to λ_i.

The iteration complexity is bounded by $O(k^2)$, based on observation of the matrix-vector multiplications from (6). ■

C. Algorithm Overview

The pseudocode for the algorithm is shown in Algorithm 1. The steps of the algorithm will be described in detail below.

Algorithm 1 A Random Algorithm for solving SDP

```
1:  Initialize $x_0 = 0$, and initialize $V^0$ randomly
2:  for t = 1,2,... do
3:      Do the gradient step from (9)
4:      Implement randomized constraint selection
5:      if An equality constraint is selected then
6:          Do the projection as in (11)
7:      else if An inequality constraint is selected then
8:          Do the projection as in (12)
9:      else
10:         Randomly select an approximate eigenvector $v_i^{t-1}$
11:         Update $v_i^{t-1}$ as in (6)
12:         Randomize $v_i^t$ as in (14)
13:         Do the projection as in (13)
14:     end if
15: end for
```

Step-size Parameters: In the algorithm, α_t and β_t are step-size parameters. For convergence purposes, the step-size parameters are chosen so that

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \beta_t = \infty, \tag{7}$$

$$\sum_{t=1}^{\infty} (\alpha_t^2 + \beta_t^2) < \infty, \quad \lim_{t\to\infty} \frac{\beta_t}{\alpha_t} = \infty \tag{8}$$

In the implementation, we used $\alpha_t = \frac{1}{t}$ and $\beta_t = \frac{1}{t^{0.51}}$. The ratio requirement $\beta_t/\alpha_t \to \infty$ enables time-scale separation argument, as in [19]. In particular, the dynamics of the approximate eigenvectors v_i^t will proceed faster than those of x_t and z_t. Thus, for large t, it can be assumed that v_i^t are approximately equal to eigenvectors of $M(z_t)$.

Gradient Step: The gradient step to optimize the objective function $c^T x$ is given by:

$$z_t = \Pi_{\mathbf{X}_0}[x_{t-1} - \alpha_t c] \tag{9}$$

where x_{t-1} is the result of $(t-1)$-th outer-loop iteration. Here, the projection operation is defined by:

$$\Pi_{\mathbf{X}_0}[y] = \arg\min_{z\in\mathbf{X}_0} \|z - y\|. \tag{10}$$

If $\mathbf{X}_0 = \mathbf{R}^n$, then the projection operation has no effect. However, for the sake of the convergence proof, $\mathbf{X}_0$ is assumed to be compact with an easily computable projection. For example, $\mathbf{X}_0$ could be taken to be a large box or large ball. This will be discussed more in Section IV.

After we obtain the intermediate point z_t, we project onto a randomly selected constraint. We first describe the projections, and then describe constraint selection methods.

Constraint Projections: Each row of the equality constraint $Ax = b$ takes the form $a_i^T x = b_i$. The projection onto the set $\mathbf{X}_i = \{x \mid a_i^T x = b\}$ is given by

$$x_t = \begin{cases} z_t - \frac{a_i^T z_t - b_i}{\|a_i\|_2^2} a_i & \text{if } a_i^T z_t \neq b_i \\ z_t & \text{otherwise} \end{cases} \tag{11}$$

Similarly, each row of the inequality constraint corresponds to a single inequality, $f_i^T x \leq g_i$. In this case, the projection onto the set $\{x \mid f_i^T x \leq g_i\}$ is given by

$$x_t = \begin{cases} z_t - \frac{f_i^T z_t - g_i}{\|f_i\|_2^2} f_i & \text{if } f_i^T z_t > g_i \\ z_t & \text{otherwise} \end{cases} \tag{12}$$

The algorithm maintains a collection of approximate eigenvectors of $M(x_t)$ given by v_i^t for $i = 1, \dots, k$. The corresponding constraint has the form $(v_i^t)^T M(x) v_i^t \leq 0$. Recall from Subsection III-A that this constraint is equivalent to a linear inequality of the form $(p^{v_i^t})^T x \leq q^{v_i^t}$. Projecting onto this constraint works the same way as (12):

$$x_t = \begin{cases} z_t - \frac{(p^{v_i^t})^T z_t - q^{v_i^t}}{\|p^{v_i^t}\|_2^2} p^{v_i^t} & \text{if } (p^{v_i^t})^T z_t - q^{v_i^t} > 0 \\ z_t & \text{otherwise} \end{cases} \tag{13}$$

Constraint Selection: The theory below ensures that the algorithm will converge as long as each constraint is selected with probability above a positive constant. Here we sketch some heuristics which speed performance in practice while satisfying the convergence requirements. For this discussion, recall that z_t is the value of the solution before projection and x_t is the value after projection.

The LMI constraint projection from (13) does not ensure that $M(x_t) \preceq 0$. Repeated application of lines 10 – 13 will converge to a solution with $M(x_t) \preceq 0$. Thus, if we find that $(v_i^t)^T M(z_t) v_i^t > 0$, we select the LMI again on the next iteration, with high probability.

To reduce effort wasted on inactive constraints, we preferentially select constraints which change the solution. For each constraint, we maintain a running average of how much the projection changed the solution, denoted by d_j^t. We select values with higher d_j^t values more often, while ensuring that all constraints are selected with positive probability.

Randomizing the eigenvectors: For the formal convergence proof below, a small amount of randomness must be added to the approximate eigenvectors, on occasion. In particular, with some small probability,

$$v_i^t = \frac{v_i^t + \epsilon w}{\|v_i^t + \epsilon w\|_2}. \tag{14}$$

Here ϵ is a small constant and w is uniformly distributed over the unit sphere.

IV. Convergence

This section proves convergence of our algorithm. In this section $\mathbf{X}_{sdp}$ will denote the set defined by the constraints of (1). We assume that the following hold.

Strict Feasibility: The inequality and LMI constraints from (1) are strictly feasible.

Compactness: The set $\mathbf{X}_0$ is compact.

Strict feasibility is a common assumptions, as it allows standard constraint qualifications [17]. In most applications, $\mathbf{X}_0 = \mathbf{R}^n$ is the most natural choice. However, the performance of the algorithm will not be altered significantly if we set $\mathbf{X}_0$ to be a very large box or ball.

To prove convergence, we show that the required assumptions from [9] are satisfied by the current problem. To describe these assumptions, we first present some notation. Collect all the variables x_t, v_i^t, and λ_i^t (along with auxiliary variables, such as d_i^t used in the constraint selection process) into a single vector w_t. Let $h_{w_t}(x)$ be the constraint applied at step t: $h_{w_t}(x) = a_i^T x - b_i$, $f_i^T x - b_i$, or $(v_i^t)^T M(x) v_i^t$. The assumptions from [9] are:

Assumption 1:

- $\mathbf{X}_0$ is non-empty, closed, and convex
- The objective function and all constraint functions are convex on $\mathbf{X}_0$
- The subgradients of the constraint functions are uniformly bounded on $\mathbf{X}_0$

Assumption 2: There exists a constant $\gamma > 0$, such that

$$dist^2(x, \mathbf{X}) \leq \gamma \mathbf{E}\Big[(h_{w_t}^+(x))^2 \mid \mathcal{F}_{t-1}\Big]$$

for all $x \in \mathbf{X_0}$ and $t \geq 1$. Here, $\mathcal{F}_{t-1}$ is the sigma-field generated by all of the variables of the algorithm, w_k, up to time $t-1$.

The main challenge is proving **Assumption 2**, which will be aided by two lemmas. The first lemma is drawn from the proof of Proposition 8 in [20].

Lemma 1: Suppose $\mathbf{X}_0$, $\mathbf{X}_1$,..., $\mathbf{X}_N$ are finitely many closed convex sets such that $\mathbf{X} = \cap_i \mathbf{X_i} \neq \emptyset$ is strictly feasible and compact. Then there exists $\xi > 0$, such that $dist^2(x, \mathbf{X}) \leq \xi \max_{i \geq 1} dist^2(x, \mathbf{X_i})$, $\forall x \in \mathbf{X}_0$.

Note that if we assume that $\mathbf{X}_0$ is compact, then we can assume without loss of generality that the set $\mathbf{X}_{sdp}$ defined by the constraints of (1) is also compact. Thus, the following lemma, equivalent to Corollary 8 of [21], can be applied.

Lemma 2: If $\mathbf{X}_{sdp}$ is a compact set defined by an LMI, $\mathbf{X}_{sdp} = \{x \mid M(x) \preceq 0\}$, then there is a constant $\kappa \geq 0$ such that for all $x \in \mathbf{R}^n$,

$$dist(x, \mathbf{X}) \leq \kappa \max_{\|v\|=1} (v^T M(x) v)^+. \tag{15}$$

*Proposition 2: **Assumption 2** holds for Algorithm 1.*

Proof: Analysis of the algorithm shows that w_t forms a Markov chain, so that w_t only depends on w_{t-1}. Thus, $\mathbf{E}\Big[(h_{w_t}^+(x))^2 \mid \mathcal{F}_{t-1}\Big] = \mathbf{E}\Big[(h_{w_t}^+(x))^2 \mid w_{t-1}\Big]$.

It was shown in [9] that **Assumption 2** holds for finite number of linear equalities and inequalities, i.e. polyhedral sets. Thus, Lemma 1 implies that without loss of generality, we can consider the case of a constraint set given by $\mathbf{X}_{sdp} = \{x \mid M(x) \preceq 0\} = \{x \mid v^T M(x) v \leq 0, \forall \|v\|_2 = 1\}$. Indeed, once **Assumption 2** is proved for LMIs, Lemma 1 can be used to prove the general case by intersecting the LMI set with the polyhedral constraint set.

Lemma 2 implies that $P(2\kappa h_{w_t}^+(x) \geq dist(x, \mathbf{X}) | w_{t-1}) > 0$ for all x and all w_{t-1}. Indeed, positivity of the event then holds by continuity of $h_{w_t}(x)$ with respect to v_i^t and the fact that all points on the sphere have non-zero density of selection, due to (14). Note that uniform randomness from (14) ensures that the probability is continuous with respect to w_{t-1} and x. Thus, compactness of $\mathbf{X}_0$ and the set of w_{t-1} will result in $P(2\kappa h_{w_t}^+(x) \geq dist(x, \mathbf{X}) | w_{t-1}) \geq \eta > 0$, for some constant η. It follows that

$$4\kappa^2 \mathbf{E}\left[(h_{w_t}^+(x))^2 | w_{t-1}\right] \geq \eta\, dist(x, \mathbf{X})^2. \tag{16}$$

Thus **Assumption 2** holds for a single LMI constraint, and as discussed above this completes the proof. ∎

We are now ready to prove convergence of the algorithm.

Theorem 1: Assume that the problem (1) has a nonempty optimal set $\mathbf{X}^*$. Then the iterates x_k generated by Algorithm 1 converge almost surely to some random point in $\mathbf{X}^*$.

Proof: In order to prove almost sure convergence, it suffices to show that our proposed algorithm satisfies the assumptions made in [9]. First, the step-size conditions on α_k from (7) satisfy the standard stochastic approximation assumptions, as required in [9].

Now **Assumption 1** will be shown. The first and second items hold by assumption. For the third item, note that all of the constraints are differentiable. Thus, the subgradients are just the gradients. Boundedness of the gradients occurs by compactness of $\mathbf{X}_0$ and the unit sphere of $\{v \mid \|v\|_2 = 1\}$.

Finally, **Assumption 2** is proved in Proposition 2. Thus, all of the assumptions required for convergence in [9] are satisfied, which completes the proof. ∎

V. Numerical Examples

In this section, we will use four examples to show our proposed algorithm's effectiveness and efficiency in solving SDP problems. Note that in our implementation, we set $\mathbf{X}_0$ to be $\mathbf{R}^n$, which does not satisfy the compactness assumption. However, our algorithm still works in practice.

Example 1: Calculating the H_∞ norm

Consider a linear system with state space representation:

$$\begin{aligned} \dot{x} &= Ax + Bw \\ z &= Cx + Dw \end{aligned} \tag{17}$$

According to [22], when A is stable, the H_∞ norm is can be computed from the following SDP:

$$\begin{aligned} \min_{P,r} \quad & r \\ s.t. \quad & \begin{bmatrix} A^T P + PA & PB & C^T \\ B^T P & -rI & D^T \\ C & D & -rI \end{bmatrix} \prec 0 \\ & P \succ 0 \end{aligned} \tag{18}$$

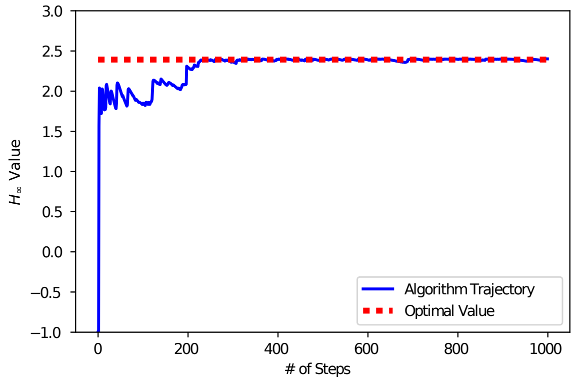

Fig. 1. H_∞ value trajectory

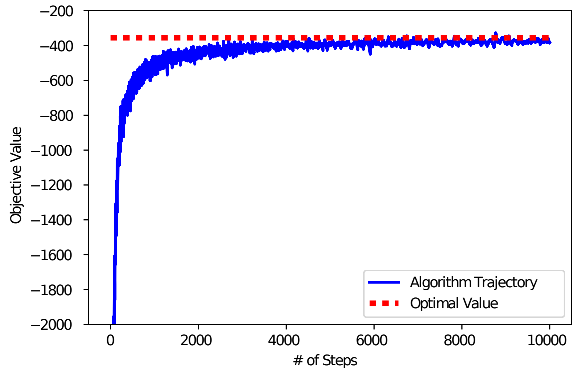

Fig. 2. Boolean Least-Squares SDP formulation objective value trajectory

We tested our proposed algorithm using the state space matrices: $A = \begin{bmatrix} -1 & 0 & 0 \\ 2 & -5 & 0 \\ 1.5 & 3 & -0.5 \end{bmatrix}$, $B = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$, $C = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$, $D = \begin{bmatrix} 2 \end{bmatrix}$.

Our proposed algorithm's objective value trajectory is shown in Figure 1. From that figure, we can see that our proposed algorithm can converge to the optimal value in a reasonable number of iterations.

Example 2: Boolean Least-Squares Consider the Boolean least-squares problem defined by:

$$\begin{array}{ll} \min & \|Ax-b\|_2^2 \\ s.t. & x_i^2 = 1, \quad i = 1, \dots, n \end{array} \tag{19}$$

According to [2], this non-convex problem is NP-hard. One efficient way to deal with such problems is to do the Lagrangian Relaxation, which produces the following SDP:

$$\begin{array}{ll} \min_{X,x} & Tr(A^TAX) - 2b^TAx + b^Tb \\ s.t. & \begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \succeq 0, \quad X_{ii} = 1, \quad i = 1, \dots, n \end{array} \tag{20}$$

Because the above SDP formulation removes the non-convex constraint for the X, it only provides the lower bound.

In our test, the 30×15 matrix A and 30×1 vector b are generated by a Gaussian distribution with zero mean and identity covariance. Our proposed algorithm gives the objective trajectory shown in Figure 2. From that figure, we can see that our proposed algorithm can reach to the optimal value, and then it stays near the optimal value. After that, we use the randomization method of [2] to obtain the 'best' feasible point with 1000 trials. The randomized feasible solution obtained from both our solver and the existing SDP solver gives an objective value of 158.93.

Example 3: Max-Cut

Consider the following problem:

$$\begin{array}{ll} \max & x^TWx \\ s.t. & x_i^2 = 1, \quad i = 1, \dots, n \end{array} \tag{21}$$

where W is a symmetric matrix with $W_{ii} = 0$ and $W_{ij} \geq 0$.

Following the same argument with Lagrangian Relaxation as in **Example 2**, Problem (21) can be reformulated as an SDP as below:

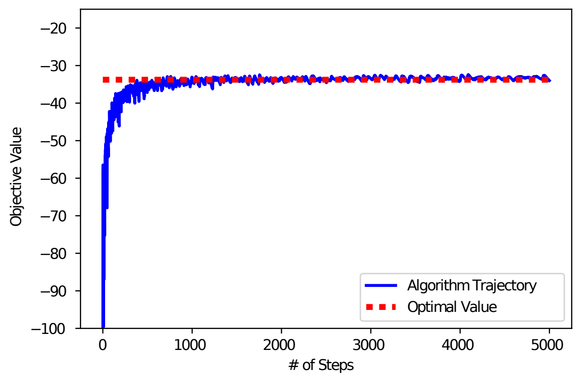

Fig. 3. Max-Cut SDP formulation objective value trajectory

$$\begin{array}{ll} \min & -Tr(WX) \\ s.t. & X \succeq 0, \quad X_{ii} = 1, \quad i = 1, \dots, n \end{array} \tag{22}$$

In our test, the 20×20 matrix W is generated by Gaussian distribution with zero mean and identity covariance. We use our proposed algorithm to solve the Problem (22), which gives the objective trajectory as shown in Figure 3. From that figure, we can see that our proposed algorithm reaches and stays near the optimal value. After that, we use the randomization method of [2] to obtain the 'best' feasible point with 1000 trials. The randomized feasible solution obtained from both our solver and the existing SDP solver gives an objective value of 352.

Example 4: Solver comparison

To compare solvers, we used random SDPs of the form:

$$\begin{array}{ll} \min\limits_{x} & c^Tx \\ s.t. & M_0 + x_1M_1 + \ldots + x_nM_n \preceq 0 \\ & \begin{bmatrix} b & x^T \\ x & I \end{bmatrix} \preceq 0 \end{array} \tag{23}$$

where $M_0, M_1, \ldots, M_n \in \mathbf{S}^n$, $x = [x_1, x_2, \ldots, x_n]^T$, b is a scalar, and I is the $n \times n$ identity matrix.

For $i = 1, \ldots, n$, the matrix M_i was generated as $M_i = H + H^T$, where H is a matrix with entries distributed according to a standard Gaussian. The matrix is set to $M_0 = \lambda I$, where λ is chosen so that the LMI is feasible for

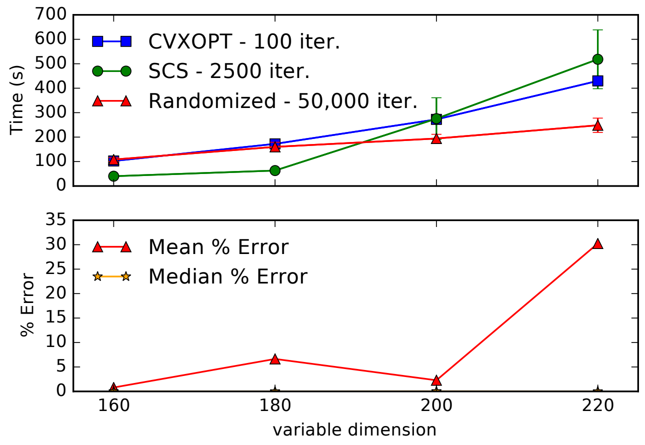

Fig. 4. Comparison of SDP solvers. The top graph shows the running time. The markers denote the mean running time of 10 instances, while the error bars denote $\pm$ one standard deviation. The bottom graph shows the percent error. Here the mean error is significantly higher than the median error, indicating that the error is dominated by relatively rare but significant errors in the solution.

a randomly generated gaussian vector $\hat{x}$. The matrix c has entries generated by a standard Gaussian. Finally, $b = 50a^2$, where a is a standard Gaussian.

Figure 4 shows the comparison of our algorithm with CVXOPT and SCS, for dimensions 160, 180, 200, and 220. For each dimension, 10 random instances were solved. Examining the figure, we see that our algorithm can perform significantly more iterations in the same running time as CVXOPT and SCS. Also, our algorithm shows better scaling with problem dimension.

However, we can see that our algorithm is occasionally prone to making large errors. Here percent error was calculated as

$$\%\text{Error} = 100 \times \frac{|val_{ours} - val_{CVXOPT}|}{|val_{CVXOPT}|}.$$

In other words, we used CVXOPT as a "ground-truth" to estimate the accuracy of our method. Note that there is a wide gap between the mean percent error and the median. Here the median value was always very low. This indicates that the algorithm was typically accurate, but a small number of inaccurate solutions produce an overall high mean error. These errors could be reduced by increasing the iterations.

VI. Conclusions and Future Work

In this paper, we propose a first-order randomization-based algorithm to solve the SDP problems. The general algorithm is simple to implement, and can flexibly incorporate various heuristic schemes to improve performance. In particular, the randomized selection schemes admit a wide variety of choices that all lead to a convergent algorithm. The algorithm was tested on a variety of practical examples, showing promising performance.

Future work will proceed along several directions. 1. Larger experiments will be done to compare with different SDP solvers. 2. Extensions to more general convex objective functions seems straightforward. 3. Here, the convergence proofs are asymptotic. Non-asymptotic convergence results would be useful for practical analysis of the method.

References

[1] C. Scherer, P. Gahinet, and M. Chilali, "Multiobjective output-feedback control via lmi optimization," *IEEE Transactions on automatic control*, vol. 42, no. 7, pp. 896–911, 1997.

[2] A. d'Aspremont and S. Boyd, "Relaxations and randomized methods for nonconvex qcqps," *EE392 Class Notes, Stanford University*, no. 1, pp. 1–16, 2003.

[3] Y. Nesterov and A. Nemirovskii, *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[4] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, 2016.

[5] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & Mathematics with Applications*, vol. 2, no. 1, pp. 17–40, 1976.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[7] B. Polyak, "Random algorithms for solving convex inequalities," *Studies in Computational Mathematics*, vol. 8, pp. 409–422, 2001.

[8] U. M. García-Palomares and F. J. González-Castaño, "Incomplete projection algorithms for solving the convex feasibility problem," *Numerical Algorithms*, vol. 18, no. 2, pp. 177–193, 1998.

[9] A. Nedić, "Random algorithms for convex minimization problems," *Mathematical programming*, vol. 129, no. 2, pp. 225–253, 2011.

[10] E. Oja and J. Karhunen, "On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix," *Journal of mathematical analysis and applications*, vol. 106, no. 1, pp. 69–84, 1985.

[11] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear matrix inequalities in system and control theory*. SIAM, 1994.

[12] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM (JACM)*, vol. 42, no. 6, pp. 1115–1145, 1995.

[13] Z.-Q. Luo, W.-K. Ma, A. M.-C. So, Y. Ye, and S. Zhang, "Semidefinite relaxation of quadratic optimization problems," *IEEE Signal Processing Magazine*, vol. 27, no. 3, pp. 20–34, 2010.

[14] F. Alizadeh, "Interior point methods in semidefinite programming with applications to combinatorial optimization," *SIAM journal on Optimization*, vol. 5, no. 1, pp. 13–51, 1995.

[15] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.

[16] Z. Wen, D. Goldfarb, and W. Yin, "Alternating direction augmented lagrangian methods for semidefinite programming," *Mathematical Programming Computation*, vol. 2, no. 3, pp. 203–230, 2010.

[17] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[18] H. Kushner and G. G. Yin, *Stochastic approximation and recursive algorithms and applications*, vol. 35. Springer Science & Business Media, 2003.

[19] V. S. Borkar, "Stochastic approximation with two time scales," *Systems & Control Letters*, vol. 29, no. 5, pp. 291–294, 1997.

[20] A. Nedić, "Random projection algorithms for convex set intersection problems," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 7655–7660, IEEE, 2010.

[21] A. Jourani and J. Ye, "Error bounds for eigenvalue and semidefinite matrix inequality systems," *Mathematical programming*, vol. 104, no. 2-3, pp. 525–540, 2005.

[22] P. Gahinet and P. Apkarian, "A linear matrix inequality approach to h_∞ control," *International journal of robust and nonlinear control*, vol. 4, no. 4, pp. 421–448, 1994.