Memory-efficient structured convex optimization via extreme point sampling

Nimita Shinde[1,2,3], Vishnu Narayanan[2], James Saunderson[3]

[1]IITB-Monash Research Academy
[2]Industrial Engineering and Operations Research, IIT Bombay
[3]Electrical and Computer Systems Engineering, Monash University

Abstract

Memory is a key computational bottleneck when solving large-scale convex optimization problems such as semidefinite programs (SDPs). In this paper, we focus on the regime in which storing an $n \times n$ matrix decision variable is prohibitive. To solve SDPs in this regime, we develop a randomized algorithm that returns a random vector whose covariance matrix is near-feasible and near-optimal for the SDP. We show how to develop such an algorithm by modifying the Frank-Wolfe algorithm to systematically replace the matrix iterates with random vectors. As an application of this approach, we show how to implement the Goemans-Williamson approximation algorithm for MaxCut using $\mathcal{O}(n)$ memory in addition to the memory required to store the problem instance. We then extend our approach to deal with a broader range of structured convex optimization problems, replacing decision variables with random extreme points of the feasible region.

1 Introduction

Semidefinite Programs (SDPs) are a class of mathematical programming problems that have a wide range of applications in areas such as control theory, statistics, signal processing, and combinatorial optimization [12, 32]. Moreover, a variety of approximation algorithms for combinatorial optimization problems involve solving a SDP relaxation and then rounding the solution to produce a feasible point with provable suboptimality guarantees. There are efficient algorithms such as interior-point methods [33], which can be used to solve SDPs. However, as the problem size increases, the memory required by these algorithms becomes a key computational bottleneck. In one regime of interest for very large-scale problems, it is not even possible to store a dense $n \times n$ decision variable in core memory.

One prominent approach to dealing with this bottleneck, pioneered by Burer and Monteiro [7], is to parameterize the positive semidefinite (PSD) decision variable as $X = UU^T$, and reformulate SDPs as nonlinear programs in the variable $U \in \mathbb{R}^{n\times r}$, where the value of r is depends on the predetermined bound on the rank of the solution and must satisfy Barvinok-Pataki bound [2, 29] for optimality guarantees. This approach has received a lot of attention (see, for example, [3, 6, 10, 19]) because it is able to resolve scalability issues with SDPs to some extent by using low-rank parameterization. However, this approach requires an *a priori* bound on the rank of an optimal solution [34]. Recently, another approach for SDPs in low memory has emerged. This involves maintaining a lower dimensional sketch of the decision variable, while preserving the convexity of the problem formulation. This approach has primarily been developed in cases where either we know an *a priori* bound on the rank of the solution [11] or the aim is to generate a low-rank approximation of the solution [35].

In this paper, we develop methods to 'solve' SDPs in low memory without prior knowledge of the rank of an optimal solution. We do not explicitly aim to represent the matrix decision

variable X, but rather to sample a zero-mean random vector with covariance X. This sampled representation of the solution only requires $\mathcal{O}(n)$ memory and is sufficient to implement rounding schemes for a number of SDP relaxations of binary optimization problems. Moreover, other succinct representations of the solution, such as low-rank approximations, can be computed in low memory by repeatedly generating sampled solutions (see Section 6).

Our aim, then, is to develop an algorithm that generates random variables with covariance that is a (near-)optimal point to a SDP. Initially, we focus on trace constrained SDPs

$$\max_X \quad g(\mathcal{B}(X)) \quad \text{subject to} \quad \begin{cases} \mathrm{Tr}(X) \leq \alpha \\ X \succeq 0, \end{cases} \tag{BoundedSDP}$$

where the objective function g is concave and smooth and $\mathcal{B}(\cdot) : \mathbb{S}^n \to \mathbb{R}^d$ is a linear mapping. This problem class was studied by Yurtsever et al. [35] and our approach is very much inspired by their work. The map $\mathcal{B}$ projects the $\binom{n+1}{2}$-dimensional variable to a much smaller d-dimensional space. One way to incorporate any additional constraints to this problem is to add a corresponding penalty term in the objective, at the expense of maintaining exactly feasible iterates (see Section 3.2). In Section 5, we discuss how to extend further our main idea to certain other constraint sets without compromising on feasibility.

1.1 Motivating Example: Maximum Cut Problem

The maximum cut problem (MaxCut) involves maximizing the quadratic function defined by the Laplacian of a graph over binary decision variables, i.e.,

$$\max_{x \in \{-1,1\}^n} \quad x^T C x \tag{MaxCut}$$

where $C = (1/4)L_G$ and L_G is the Laplacian of a graph. Note that the cost matrix C is a symmetric diagonally dominant matrix with positive entries on the diagonal. In a celebrated result, Goemans and Williamson [14] developed a α_{GW}-approximation algorithm (with $\alpha_{GW} \approx 0.878$) that involves solving the SDP relaxation

$$\max_{\mathrm{diag}(X)=\mathbb{1}, X \succeq 0} \quad \langle C, X \rangle \tag{MaxCut-SDP}$$

followed by a randomized rounding scheme. If $X^\star$ is an optimal solution of (MaxCut-SDP), the rounding scheme involves sampling a zero-mean Gaussian vector z with covariance $X^\star$, so that the binary vector $\mathrm{sign}(z)$ is a random feasible point to MaxCut that achieves the stated approximation guarantee in expectation. To implement this rounding scheme, there is no need to explicitly compute $X^\star$; instead it is enough to construct a zero-mean random (Gaussian) vector with covariance $X^\star$. This observation is a key motivation for our notion of sampled solutions for SDPs.

Following [36], we define the working memory of an algorithm as follows.

Definition 1.1. The *working memory* of an algorithm is defined as the total memory utilized by the algorithm apart from the memory required to represent the problem instance.

If we have an algorithm to solve (MaxCut-SDP) that can track such samples rather than the full decision variable, we can potentially implement the Goemans-Williamson method using $\mathcal{O}(n)$ working memory. One of the main contributions of this paper is to show that this in fact possible (see Algorithm 3).

1.2 Our Contributions

We now summarize the key contributions of the paper.

Sample-based solutions to convex programs A key conceptual contribution of this paper is to propose the idea of sample-based solutions to convex programs. We consider two approaches:

- Gaussian samples: In this case, the aim is to represent the positive semidefinite solution X of a SDP via a zero mean Gaussian vector $z \sim \mathcal{N}(0, X)$ such that its covariance is X.
- Extreme-point samples: In this case, the aim is to represent the solution x of a convex program via a random extreme point of the feasible region such that its expected value is x.

Generating Gaussian sample-based solution to SDPs Using an algorithmic framework based on the Frank-Wolfe algorithm, we show that it is possible to compute an ϵ-optimal Gaussian sample-based solution of (BoundedSDP) and a near-feasible, near-optimal Gaussian sample-based solution to SDP with d linear equality constraints and bounded feasible region. The working memory of our algorithm is $\mathcal{O}(n+d)$ and it generates a solution with provable optimality guarantees after $\mathcal{O}\left(n^2/\epsilon^2\right)$ iterations (see Lemma 3.2).

Approximation algorithm for MaxCut For MaxCut, we provide an implementation of Goemans-Williamson rounding method that results in a $(1-\epsilon)\alpha_{GW}$-approximate solution (with $\alpha_{GW} \approx 0.878$) to MaxCut that requires additional storage of at most $3n$ numbers. This result Theorem 4.1 in Section 4 is stated in a less detailed form below.

Theorem 1.1. *Given $\epsilon \in (0,1)$ and a diagonally dominant cost matrix C, there exists a polynomial time $\mathcal{O}\left(\frac{n^3}{\epsilon^3}\log(2n)\log\left(\frac{2n}{\epsilon}\right) \times \text{mvc}\right)$ randomized algorithm, where* mvc *is the complexity of matrix-vector multiplication with C, that generates a random binary vector $w \in \{-1,1\}^n$ which satisfies*

$$\alpha_{GW}(1-\epsilon)\text{opt} \leq \mathbb{E}[w^T C w] \leq \text{opt}, \tag{1.1}$$

where opt *is the maximum of $w^T C w$ over the set $w \in \{-1,1\}^n$. The working memory of the algorithm is at most $3n$ numbers.*

A key conceptual difference between our approach and existing factorization or sketching methods is that no *a priori* bound on the rank of the optimal solution is required. Indeed, in the case of (MaxCut-SDP), it is well-known that the optimal solution can have rank $\Theta\left(\sqrt{n}\right)$ [2, 29].

Generating extreme-point sample-based solution to convex programs For a convex optimization problem with compact feasible region, if the extreme points of the feasible region can be represented in low memory, then we can track its solution in low memory provided that we can solve the linear optimization subproblem over the feasible region in low memory. This allows us to move to a more general setting where the decision variable need not be a positive semidefinite matrix. We provide a modified Frank-Wolfe algorithm (see Algorithm 4) whose output is a random extreme point such that it satisfies the optimality bounds for constrained convex program in expectation.

1.3 Related Work on Low Memory Algorithms for SDP

In this subsection, we briefly summarize key ideas in the literature related to low memory algorithms for SDP with d linear equality constraints and bounded feasible region. One approach is to replace the PSD matrix by a low-rank factorization and use nonlinear programming techniques to compute the solution of the resulting nonconvex problem. This technique was pioneered by Burer and Monteiro [7]. The factorization sacrifices convexity and its associated optimality guarantees and typical numerical algorithms are only able to generate first- or

second-order critical points. Nevertheless, Boumal, Voroninski, and Bandeira [5] showed that if the constraint set is a smooth manifold and the rank r of the factorization is set to at most r with $r(r+1) \geq 2d$, then any second-order critical point for SDP is a global optimum. This bound is a consequence of Barvinok-Pataki bound [2, 29] which states that SDP with d linear equality constraints and a bounded feasible region admits a global optimum with rank r that satisfies $r(r+1) \leq 2d$. Using a second-order method [19] or a Riemannian gradient descent algorithm [6], it is possible to compute an ϵ-optimal solution to the factorized problem that satisfies the Barvinok-Pataki bound, and requires $\mathcal{O}(n\sqrt{d})$ working memory. On the other hand, Waldspurger and Waters [34] showed that unless r is at least as large as $\sqrt{2d}$, a typical numerical algorithm can converge to nonoptimal critical points for the resulting nonconvex problem. This method not only requires an *a priori* knowledge of the rank of the solution but it is also requires $\Omega(n\sqrt{d})$ working memory to generate a solution with provable optimality guarantees.

A recent approach to solving linearly constrained SDPs in low memory involves sketching the decision variable to a low dimensional subspace and using first-order algorithms to compute a solution of the optimization problem in the space of the sketched decision variable. Unlike factorization approaches that break the convexity of the linearly constrained SDP, sketching the variable preserves the convexity of the problem formulation. These methods are based on techniques for sketching low rank matrices. For instance, Tropp et al. [31] provide an algorithmically simple sketching technique to compute a low-rank approximation of a matrix $A \in \mathbb{R}^{n\times n}$ by exploiting the spectral decay of the matrix. Their algorithm preserves the positive semidefiniteness of the decision variable.

Such sketching and reconstruction techniques were extended by Yurtsever et al. [35] to generate an approximate rank-r factorization of a solution to a smooth convex optimization problem with bounded nuclear norm constraint. They do so by sketching the decision variable to a lower dimension and using the Frank-Wolfe algorithm to track the sketched variable. With the sketched variable, the working memory required for their method is $\Theta(d+nr)$. Although the sketch used in [35] is different from the sample-based solutions in this paper, the algorithmic architecture of our approach is inspired by [35].

In their recent work, Yurtsever et al. [36] extend the approach from [35] to deal with SDPs with d linear equality constraints. They provide a polynomial-time randomized sketching algorithm that, with high probability, computes a rank-r approximation of a near-feasible solution to SDP with d linear equality constraints. The working memory required to compute this solution via sketching is $\mathcal{O}(d+rn/\zeta)$, i.e., it is linear in number of equality constraints and the rank of the computed solution. The nonnegative quantity $\zeta \in (0,1)$ controls how close the rank-r approximation X_r produced by the algorithm is to the best rank-r approximation $\widehat{X}$ of the solution X. More precisely, with high probability these quantities satisfy $\|X - X_r\|_\star \leq (1+\zeta)\|X - \widehat{X}\|_\star$, where $\|\cdot\|_\star$ is the nuclear norm. We will discuss the similarities and differences between our work and [36] in Section 7.

Alternatively, Ding et al. [11] compute a low-rank solution to an SDP with linear equality constraints by first approximately finding the subspace in which the solution lies. This subspace is computed by finding the null space of the dual slack variable. By restricting the search space of the primal solution to a smaller subspace, they restrict the memory required. The dual problem is solved only approximately, with the assumption that the rank of the primal optimal solution satisfies the Barvinok–Pataki bound [2, 29], which results in some error in the computation of the subspace in which the primal solution lies. The working memory for their method is then $\Theta(d+nr)$, where r is the rank of an optimal solution to SDP and satisfies the Barvinok-Pataki bound. Table 1 shows the comparison of memory required by the above methods with the memory requirement of Algorithm 2. Note that the methods given in the literature compute a rank-r solution to SDP with d linear equality constraints while Algorithm 2 generates random samples that have covariance equal to a near-feasible, near-optimal solution to the SDP.

Reference	Working Memory Required
Tropp et al. [31]	$\Theta(d+nr)$
Yurtsever et al. [36]	$\mathcal{O}(d+rn/\zeta)$
Ding et al. [11]	$\Theta(d+nr)$
Algorithm 2	$\mathcal{O}(d+n)$

Table 1: Working memory requirement to compute the solution to SDP with d linear equality constraints. The first three methods in the table compute a rank-r approximate solution and Algorithm 2 from this paper generates a sample-based solution

A widely studied special case of SDP is (MaxCut-SDP). Since (MaxCut-SDP) has n linear equality constraints, the Barvinok-Pataki bound [2, 29] states that the rank of the optimal solution could be as high as $\Omega(\sqrt{n})$. As such, the algorithms for solving (MaxCut-SDP) could require $\Omega(n^{1.5})$ working memory. One such algorithm is given by Klein and Lu [20] which generates a rank-1 update to the intermediate solution of (MaxCut-SDP) at each iteration. The number of iterations required for convergence, and thus the rank of the intermediate solution could be as large as n. By restricting the rank of the solution to $\mathcal{O}(\sqrt{n})$, Klein and Lu [21] restrict the working memory to be $\mathcal{O}\left(n^{1.5}\right)$. An improvement on this memory requirement is provided by Yurtsever et al. [36]. Their sketching method generates, with high probability, a rank-1 approximation zz^T of a near-feasible, ϵ-optimal solution X to (MaxCut-SDP) using $\mathcal{O}(n/\zeta)$ working memory such that for some $\zeta \in (0,1)$, $\|X - zz^T\|_\star \leq (1+\zeta)\|X - \widehat{X}\|_\star$, where $\widehat{X}$ is the best rank-1 approximation of X and $\|\cdot\|_\star$ is the nuclear norm. Note that, the memory required is linear in n, however, it has dependence on the approximation parameter ζ.

Our sampling technique requires $\mathcal{O}(n)$ memory as we only aim to represent the samples of a near-optimal solution. This eliminates the dependency on the rank r of the low rank approximation of a near-optimal solution or the accuracy to which the approximation is computed.

1.4 Notations

The inner product $\langle A, B\rangle = \text{Tr}\left(A^T B\right)$ denotes the matrix inner product and $\|A\|_F^2 = \langle A, A\rangle$ is the Frobenius norm. Unless otherwise specified, $\|\cdot\|$ represents Euclidean norm for vectors. For a matrix X, $\text{diag}(X)$ represents a vector of the diagonal entries of matrix X. For a vector x, $\text{diag}^*(x)$ represents a diagonal matrix with the entries of x on the diagonal. The vector $\mathbb{1}$ has each element value equal to one. The notation $\nabla g(\cdot)$ is used to denote the gradient and $\nabla^2 g(\cdot)$ is used to denote the Hessian of a twice differentiable function g. We use $\mathcal{A}(\cdot) : \mathbb{S}^n \to \mathbb{R}^d$ to denote a linear mapping of a symmetric $n \times n$ matrix to a d-dimensional space and $\mathcal{A}^*(\cdot) : \mathbb{R}^d \to \mathbb{S}^n$ to denote its adjoint. The notation $\succeq$ denotes the semidefinite order. The notation $\lambda_{\max}(\cdot)$ is used to denote the largest eigenvalue of a matrix. The notations $\mathcal{O}, \Omega, \Theta$ have the usual computer science complexity interpretation.

1.5 Outline

The paper is organized as follows. In Section 2, we discuss standard results related to the Frank-Wolfe algorithm. These form the basis of algorithms and analysis discussed later. In Section 3, we present the idea of sampled solutions to SDPs, and provide a modified version of Frank-Wolfe that generates such sampled solutions. In Section 4, we apply our algorithm to generate sampled solutions to (MaxCut-SDP). We also show how to round this sampled solution to give an implementation of the Goemans-Williamson method that requires $\mathcal{O}(n)$ working memory. In Section 5 we discuss an extension of the idea of sampled solutions to SDPs to a more general setting in which the feasible solution is no longer required to be a trace constrained PSD matrix. We propose a modification of the Frank-Wolfe algorithm that generates a random extreme point of the feasible region, such that the expectation of the generated random

solution is near-optimal. In Section 6, we discuss how to obtain other approximations of the solution of structured convex programs by combining our sampled representations with streaming algorithms. Section 7 discusses further possible extensions of our work and briefly presents preliminary numerical experiments.

2 Preliminaries

Consider the following optimization problem,

$$\max_{x \in \mathcal{S}} \quad g(\mathcal{B}(x)) = \max_{v \in \mathcal{B}(\mathcal{S})} \quad g(v) \tag{Constrained-OPT}$$

where $\mathcal{B}(\cdot) : \mathbb{R}^m \to \mathbb{R}^d$ is a linear map, $g(\cdot)$ is a smooth, concave function, and $\mathcal{S} \subseteq \mathbb{R}^m$ is a compact, convex set. The variable $v = \mathcal{B}(x)$ is said to be a 'projection' of the 'lifted' decision variable x of (Constrained-OPT) and the map $\mathcal{B}(\cdot)$ can be interpreted as linear measurements of the decision variable. In this section, we briefly review the Frank-Wolfe algorithm [13], which is central to our algorithmic approach. We recall the modification done to the steps of Frank-Wolfe to adapt it to problems of type (Constrained-OPT) by Yurtsever et al. [35] and give subsequent convergence results. Moreover, we also discuss (BoundedSDP), which is a special case of (Constrained-OPT).

2.1 An Approximate Frank-Wolfe Algorithm

The Frank-Wolfe algorithm [13] is an iterative algorithm for convex optimization over a compact, convex feasible region that solves a linear optimization problem at each iterate. The algorithm then, in each iteration, takes a fractional step towards a maximizer of the linear optimization subproblem which can be taken to be an extreme point of the feasible set. Frank-Wolfe is an example of a 'projection-free' method since the fractional step ensures that the next iterate is feasible, eliminating the need to project onto the feasible region.

An approximate Frank-Wolfe algorithm (see, e.g., [17]) computes an ϵ-optimal solution of (Constrained-OPT) by solving the linear subproblem approximately (with some finite additive error) at each iteration.

Definition 2.1. A feasible point $\bar{x} \in \mathcal{S}$ is called ϵ-optimal for the optimization problem $\max_{x \in \mathcal{S}} f(x)$ if $f(\bar{x}) \geq \max_{x \in \mathcal{S}} f(x) - \epsilon$.

For a concave objective function g and a convex compact feasible domain $\mathcal{S}$, Algorithm 1 details an approximate Frank-Wolfe algorithm to compute an ϵ-optimal solution to (Constrained-OPT).

In Algorithm 1, we assume that at each iteration, the linear optimization subproblem `LMO` is solved by a randomized method that succeeds with probability at least $1 - p$. Moreover, we assume the randomness in this subproblem is independent across function calls. In Algorithm 1, we not only keep track of the 'lifted' iterate $x_t \in \mathbb{R}^n$ (decision variable) at each iteration but also the 'projected' iterate $v_t \in \mathbb{R}^d$. The knowledge of these projected iterates v_t is enough to compute the gradient of the objective function, and hence to compute the update direction. Thus, for a high dimensional decision variable, we can simply track the projected iterates v_t to track the improvement in objective function value at each iterate. This is a key observation from the work of Yurtsever et al. [35] that is also crucial here.

Curvature constant. The curvature constant C_g is a measure of nonlinearity of the objective function g over the feasible set $\mathcal{S}$. It is defined (see, e.g., [17]) as,

$$C_g = \sup_{\substack{x,h \in \mathcal{S}, \\ y = v + \gamma(h-v), \\ \gamma \in [0,1]}} -\frac{2}{\gamma^2} \left(g(y) - g(v) - \nabla g(v)^T (y - v) \right).$$

Algorithm 1: Frank-Wolfe Algorithm with Approximate Solution to Subproblem (2.1)

Input : Stopping criteria ϵ, accuracy parameter η, probability p for subproblem (2.1), upper bound C_g^u on the curvature constant

Output: ϵ-optimal maximizer x_t of $g(\mathcal{B}(x))$, $v_t = \mathcal{B}(x_t)$

Function `FWApproxSubprob`**:**
  Select initial point $x_0 \in \mathcal{S}$ and set $v_0 = \mathcal{B}(x_0)$
  $t \leftarrow 0$
  $\gamma \leftarrow \frac{2}{t+2}$
  $(h_t, q_t) \leftarrow \texttt{LMO}(\mathcal{B}^*(\nabla g(v_t)), \frac{1}{2}\eta\gamma C_g^u, p)$
  while $\langle q_t - v_t, \nabla g(v_t)\rangle > \epsilon$ **do**
    $(x_{t+1}, v_{t+1}) \leftarrow \texttt{UpdateVariable}(x_t, v_t, h_t, q_t, \gamma)$
    $t \leftarrow t + 1$
    $\gamma \leftarrow \frac{2}{t+2}$
    $(h_t, q_t) \leftarrow \texttt{LMO}(\mathcal{B}^*(\nabla g(v_t)), \frac{1}{2}\eta\gamma C_g^u, p)$
  end
return (x_t, v_t)
Function `LMO`$(J, \delta,\ p)$**:**
  Find $h \in \mathcal{S}$ such that with probability at least $1 - p$, $\langle h, J\rangle \geq \max_{s\in\mathcal{S}} \langle s, J\rangle - \delta$
  $q = \mathcal{B}(h)$
return (h, q)
Function `UpdateVariable`(x, v, h, q, γ)**:**
  $x \leftarrow (1 - \gamma)x + \gamma h$
  $v \leftarrow (1 - \gamma)v + \gamma q$
return (x, v)

The value of the curvature constant provides insight into the deviation of the linearization of the function g from the actual function value. If g is twice-differentiable, then we often use the upper bound, $C_g \leq C_g^u = \lambda_{\max}(-\nabla^2 g)\text{diam}(\mathcal{S})^2$, where $\text{diam}(\mathcal{S})$ is the Euclidean diameter of the set $\mathcal{S}$.

Approximate solution to subproblem in `LMO`. The subroutine `LMO` solves the linear maximization problem

$$\max_{h\in\mathcal{S}} \quad \langle h, \mathcal{B}^*(\nabla g(v))\rangle \tag{2.1}$$

approximately at each iteration t. The approximate maximizer h_t is computed such that with probability at least $1-p$, the additive error in the function value at h_t is at most $\frac{1}{2}\eta\gamma C_g^u$, where $\eta \geq 0$ is a fixed accuracy parameter and C_g^u is an upper bound on the curvature constant of g. Initially, this margin of error is high; but with each iteration we solve the subproblem (2.1) to a higher accuracy. When $p = 0$, Algorithm 1 is equivalent to the Frank-Wolfe algorithm given by Jaggi [17]. By setting $p > 0$, we require the subroutine to adhere to the desired accuracy with some probability of failure.

Stopping criterion. Algorithm 1 terminates when

$$\langle q_t - v_t, \nabla g(v_t)\rangle \leq \epsilon$$

is satisfied. When this condition holds, it follows that $g(v_t) \geq g(v^\star) - \epsilon$, i.e., v_t is an ϵ-optimal solution of (Constrained-OPT) [17].

Convergence of the Frank-Wolfe algorithm.

Theorem 2.1. *Let $g : \mathbb{R}^d \to \mathbb{R}$ be a concave and differentiable function and $x^\star$ an optimal solution of (Constrained-OPT). If C_g^u is an upper bound on the curvature constant of g, and $\eta \geq 0$ is the accuracy parameter for subproblem (2.1), then x_t, the t-th iterate of Algorithm 1, satisfies*

$$- g(\mathcal{B}(x_t)) + g(\mathcal{B}(x^\star)) \leq \frac{2C_g^u(1+\eta)}{t+2} \tag{2.2}$$

with probability $(1-p)^t \geq 1 - tp$.

Proof. The results follows from [17, Theorem 1] when the subproblem (2.1) is solved to an accuracy of $\frac{1}{2}\eta\gamma C_g^u$ with probability $1-p$. □

Thus, after $t = \frac{2C_g^u(1+\eta)}{\epsilon} - 2$ iterations, the solution x_t satisfies

$$g(\mathcal{B}(x_t)) \geq g(\mathcal{B}(x^\star)) - \epsilon \tag{2.3}$$

with probability at least $1-tp$. So, we can now compute an ϵ-optimal solution to (Constrained-OPT) in $t \sim \mathcal{O}(C_g^u/\epsilon)$ iterations with probability at least $1 - tp$.

In Algorithm 1, we solve the subproblem (2.1) approximately to find an update direction that is an extreme point of the set $\mathcal{S}$. If the following conditions on the constraint set $\mathcal{S}$ are satisfied, then we potentially generate the sampled representation of the solution to Problem (Constrained-OPT) in low memory as illustrated in Section 2.2.

LowMemoryComputations The image of the linear map $\mathcal{B}$ is low dimensional and the subproblem (2.1) can be solved in low memory.

LowMemoryExtremePoints The extreme points of the feasible set $\mathcal{S}$ can be represented in low memory.

2.2 SDP with Bounded Trace Constraint

For most of the paper, our focus is on semidefinite programming problems. As such, we briefly focus on (BoundedSDP), a special SDP that has a trace constrained feasible set and a concave objective function g. The extreme points of the feasible set $\mathcal{S} = \{X \succeq 0 : \mathrm{Tr}(X) \leq \alpha\}$ are rank-1 PSD matrices and can be represented in low memory. Moreover, a solution of subproblem (2.1) at iterate t is an eigenvector of the matrix $J = \mathcal{B}^*(\nabla g(\mathcal{B}(X_t)))$ corresponding to its largest eigenvalue. To solve this subproblem approximately, we find w_t such that $\alpha w_t^T J w_t \geq \alpha\lambda_{\max}(J) - \frac{1}{2}\eta\gamma C_g^u$ and then select the update direction

$$H_t = \begin{cases} \alpha w_t w_t^T, & \text{if } w_t^T J w_t \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

The computational complexity of each iteration depends on computing a rank-1 matrix that satisfies this inequality, i.e., solving an approximate eigenvalue problem. We use power method with random start to compute λ_t and the unit vector w_t. Kuczyński and Woźniakowski [22] provide error bounds for power method when the input matrix is PSD. In Lemma 2.1, we restate their result so it applies to the largest eigenvalue of a symmetric matrix J.

Lemma 2.1. *Let $J \in \mathbb{S}^n$ and $\lambda = \max_i |\lambda_i(J)|$. For $\delta \geq 0$, $p \in [0,1)$ and $\alpha \geq 0$, the power method with random start computes an unit vector w that satisfies*

$$\alpha w^T J w \geq \alpha\lambda_{\max}(J) - \delta \tag{2.5}$$

with probability at least $1-p$ after $k \geq \frac{\lambda\alpha}{\delta}\log\left(\frac{n}{p^2}\right)$ iterations. Each iteration of the power method consists of a matrix-vector multiplication with J and the working memory is n numbers.

Note that, for (BoundedSDP), it it sufficient to store the input parameters, the map $\mathcal{B}(\cdot)$ and the rank-1 updates. If we have an access to a black box performing matrix-vector multiplications with J, the Frank-Wolfe algorithm, when applied to (BoundedSDP), has working memory bounded by $\mathcal{O}(n+d)$ at each step.

3 Frank-Wolfe Algorithm with Gaussian Sampling

We now explain how to modify Algorithm 1, when applied to (BoundedSDP), to replace the matrix-valued iterates with Gaussian vectors such that their covariance is equal to the iterate value. We then show how to apply this approach to more general SDPs by incorporating the constraints into the objective with a penalty.

3.1 Idea of Gaussian Sampling

Consider the Frank-Wolfe update for (BoundedSDP) at iterate t, i.e., $X_{t+1} = (1-\gamma_t)X_t + \gamma_t H_t$. Assume that, at iterate t, we have a zero-mean random vector z_t with covariance X_t. The update direction $H_t = w_t w_t^T$ has rank one, so if $\zeta \sim \mathcal{N}(0,1)$, then $\zeta w_t \sim \mathcal{N}(0, H_t)$. Now, if we define $z_{t+1} = \sqrt{1-\gamma_t} z_t + \sqrt{\gamma_t}\zeta w_t$, then $\mathbb{E}\left[z_{t+1} z_{t+1}^T\right] = (1-\gamma_t)X_t + \gamma_t H_t = X_{t+1}$.

A nonnegative weighted sum of the samples then gives rise to nonnegative weighted sum of their covariance matrices. Furthermore, we can generate a sample at the next iterate in $\mathcal{O}(n)$ memory when the update has rank at most one.

3.1.1 Frank-Wolfe algorithm with Gaussian sampling

Algorithm 2 incorporates Gaussian sampling into Algorithm 1 by replacing the matrix variables by the sampled representation. The main algorithm is similar to that in Algorithm 1. The difference lies in the functions `LMO` and `UpdateVariable`:

- `LMO`: In Algorithm 1, this function simply computes the update direction by computing the approximate maximizer of $\langle s, \mathcal{B}^*(\nabla g(v))\rangle$ over the feasible set with additive error at most $\frac{1}{2}\eta\gamma C_g^u$. In Algorithm 2, we replace this step by computing the vector w such that $w_t w_t^T$ solves the subproblem (2.1) approximately and the update satisfies (2.4).
- `UpdateVariable`: In Algorithm 1, we update the decision variable x and its projection $v = \mathcal{B}(x)$ at every iteration. In Algorithm 2, we only track the Gaussian sample $z \sim \mathcal{N}(0, X)$ and $v = \mathcal{B}(X)$ which requires only $\mathcal{O}(n+d)$ memory instead of $\mathcal{O}(n^2+d)$.

Algorithm 2 gives a detailed description of the sampled modification of Frank-Wolfe applied to (BoundedSDP). The convergence rate of the Frank-Wolfe algorithm given in Theorem 2.1 applies even with the incorporation of Gaussian sampling, i.e., Algorithm 2 converges to z_t such that $\mathbb{E}\left[z_t z_t^T\right]$ is ϵ-optimal for (BoundedSDP) after $t \sim \mathcal{O}\left(\frac{C_g^u}{\epsilon}\right)$ iterations. We summarize the result of Algorithm 2 applied to (BoundedSDP) in Proposition 3.1.

Proposition 3.1. *The output of Algorithm 2 is a zero-mean Gaussian random vector $\widehat{z}_\epsilon$ with covariance $\widehat{X}_\epsilon$, where $\widehat{X}_\epsilon$ is an ϵ-optimal solution of (BoundedSDP). The working memory of the algorithm is $\mathcal{O}(n+d)$, where the d is the dimension of the image of $\mathcal{B}$.*

3.2 SDP with Linear Equality Constraints

Consider an SDP with linear objective function and linear equality constraints written as,

$$\max_X \ \langle C, X\rangle \quad \text{subject to} \quad \begin{cases} \mathcal{A}(X) = b \\ X \succeq 0, \end{cases} \tag{SDP}$$

Algorithm 2: Frank-Wolfe Algorithm with Gaussian Sampling

Input : Input data for (BoundedSDP), stopping criteria ϵ, accuracy parameter η, probability p for subproblem (2.1), upper bound C_g^u on the curvature constant

Output: Sample $z \sim \mathcal{N}(0, \widehat{X}_\epsilon)$ and $v = \mathcal{B}(\widehat{X}_\epsilon)$ such that $\widehat{X}_\epsilon$ is an ϵ-optimal solution of (BoundedSDP)

Function `FWGaussian`:
  Select initial point $X_0 \in \mathcal{S}$; $X_0 \leftarrow \frac{\alpha}{n} I$(say) and set $v_0 \leftarrow \mathcal{B}(X_0)$
  Sample $z_0 \sim \mathcal{N}(0, X_0)$
  $t \leftarrow 0$
  $\gamma \leftarrow 2/(t+2)$
  $(w_t, q_t) \leftarrow \texttt{LMO}(\mathcal{B}^*(\nabla g(v_t)), \frac{1}{2}\eta\gamma C_g^u, p)$
  while $\langle q_t - v_t, \nabla g(v_t)\rangle > \epsilon$ **do**
    $(z_{t+1}, v_{t+1}) \leftarrow \texttt{UpdateVariable}(z_t, v_t, w_t, q_t, \gamma)$
    $t \leftarrow t+1$
    $\gamma \leftarrow 2/(t+2)$
    $(w_t, q_t) \leftarrow \texttt{LMO}(\mathcal{B}^*(\nabla g(v_t)), \frac{1}{2}\eta\gamma C_g^u, p)$
  end
return (z_t, v_t)
Function `LMO`$(J,\ \delta,\ p)$**:**
  Find a unit vector w such that with probability at least $1-p$, $\alpha\langle ww^T, J\rangle \geq \max_{d\in\mathcal{S}} \alpha\langle d, J\rangle - \delta$
  $\lambda \leftarrow \langle ww^T, J\rangle$
  if $\lambda \geq 0$ **then**
    $q \leftarrow \mathcal{B}(\alpha ww^T)$
  else
    $q \leftarrow 0$
    $w \leftarrow 0$
  end
return (w, q)
Function `UpdateVariable`(z, v, w, q, γ)**:**
  $z \leftarrow (\sqrt{1-\gamma})z + \sqrt{\gamma} w\zeta$ where $\zeta \sim \mathcal{N}(0,1)$
  $v \leftarrow (1-\gamma)v + \gamma q$
return (z, v)

where $\mathcal{A}(\cdot) : \mathbb{S}^n \to \mathbb{R}^d$ is a linear map. We assume that the feasible region is bounded. In Section 2.2, we saw that when Algorithm 1 is applied to (BoundedSDP), the rank of the update variable is at most one. And in Section 3.1, we saw that in this case we can update and track the change in a Gaussian sample representing the matrix-valued decision variable without explicitly computing the matrix at intermediate steps. Because of the simplicity of solving subproblem (2.1) and updating the samples for trace constrained problems, we penalize the linear equality constraints in (SDP) with a smooth penalty function so that the feasible domain is reduced to the one in (BoundedSDP). By penalizing the constraints, we can only generate a near-feasible point to (BoundedSDP), however, the extreme points of the modified constraint set now have a concise representation. In the specific case of (MaxCut-SDP), it is possible to generate a feasible solution with relative error bounds on the objective function value if we know a near-feasible solution for the problem with infeasibility error bounded by $\|\mathcal{A}(X) - b\|_\infty$. This motivates us to use a penalty function (3.1) that approximates $\|\mathcal{A}(X) - b\|_\infty$.

Penalty function. For $M > 0$, let $\phi_M(\cdot) : \mathbb{R}^d \to \mathbb{R}$ be defined by

$$\phi_M(v) = \frac{1}{M} \log \left(\sum_{i=1}^{d} e^{M(v_i)} + \sum_{i=1}^{d} e^{M(-v_i)} \right). \tag{3.1}$$

This function, also known as LogSumExp (LSE), is a smoothed approximation of $\|v\|_\infty$ as the next well-known proposition shows.

Proposition 3.2 (Bound on penalty [15])**.** *If $\phi_M(\cdot)$ is defined as in (3.1), then*

$$\|v\|_\infty \leq \phi_M(v) \leq \frac{\log(2d)}{M} + \|v\|_\infty. \tag{3.2}$$

We add this penalty term to the objective function of (SDP) to penalize the equality constraints and then solve the problem

$$\max_X \ \langle C, X \rangle - \beta \phi_M(\mathcal{A}(X) - b) \quad \text{subject to} \quad \begin{cases} \operatorname{Tr}(X) \leq \alpha, \\ X \succeq 0, \end{cases} \tag{SDP-LSE}$$

where M and β are positive constants to be chosen later, and α is chosen such that $X \succeq 0$ and $\mathcal{A}(X) = b$ implies that $\operatorname{Tr}(X) \leq \alpha$. This is possible because the feasible region is assumed to be bounded.

A similar approximation and penalty technique is used by Hazan [16] to compute an approximate solution of a feasibility problem with linear inequality constraints. The objective function in this case is simply $\phi_M(\mathcal{A}(X) - b)$ and the optimal objective value is zero.

Let $(u, v) = \mathcal{B}(X) = (\langle C, X \rangle, \mathcal{A}(X))$ so that the objective function of (SDP-LSE) can be expressed as $g(u, v) = u - \beta \phi_M(v - b)$. This has the same structure as (BoundedSDP). We can now use Algorithm 2 to compute an ϵ-optimal solution $\widehat{X}_\epsilon$ of (SDP-LSE). Moreover, we will show (in Lemma 3.2) that by choosing the parameters M and β approximately, $\widehat{X}_\epsilon$ is also a near-feasible, near-optimal solution of (SDP). The convergence rate and the bounds on the infeasibility and objective function value at $\widehat{X}_\epsilon$ for (SDP) are given in Lemma 3.2.

3.2.1 Convergence of the Frank-Wolfe algorithm

We have seen that Theorem 2.1 states the convergence result for Algorithm 1 when applied to (Constrained-OPT). The algorithm converges to an ϵ-optimal solution after $\mathcal{O}\left(\frac{C_g^u}{\epsilon}\right)$ iterations. Moreover, this convergence result also holds for Algorithm 2 when applied to (SDP-LSE). We now determine an upper bound on the curvature constant C_g for (SDP-LSE).

Lemma 3.1. *An upper bound on the curvature constant C_g of the concave, smooth function $g(\mathcal{B}(X)) = \langle C, X \rangle - \beta \phi_M(\mathcal{A}(X) - b)$ over the compact, convex set $\mathcal{S} = \{X \succeq 0 : \operatorname{Tr}(X) \leq \alpha\}$ is*

$$C_g \leq \beta \omega M \alpha^2,$$

where $\omega = \max_i \lambda_{\max}(A_i)$.

Proof. Let $f(X) = -\phi_M(\mathcal{A}(X) - b)$ and let $\mathcal{S}_f = \{X \succeq 0 : \operatorname{Tr}(X) \leq 1\}$. An upper bound on the curvature constant C_f of f over $\mathcal{S}_f$, given by Hazan [16] is

$$C_f \leq \lambda_{\max}(-\nabla^2 f) \operatorname{diam}(\mathcal{S}_f)^2 \leq \omega M. \tag{3.3}$$

Note that $\lambda_{\max}(-\nabla^2 g) = \beta \lambda_{\max}(-\nabla^2 f)$ and $\operatorname{diam}(\mathcal{S})^2 = \alpha^2 \operatorname{diam}(\mathcal{S}_f)^2$. Hence, an upper bound on the curvature constant of g over $\mathcal{S}$ is

$$C_g \leq \lambda_{\max}(-\nabla^2 g) \operatorname{diam}(\mathcal{S})^2 = \beta \alpha^2 \lambda_{\max}(-\nabla^2 f) \operatorname{diam}(\mathcal{S}_f)^2 \leq \beta \omega M \alpha^2. \qquad \square$$

3.2.2 Optimality and feasibility results for (SDP)

Given $\widehat{X}_\epsilon$, an ϵ-optimal solution to (SDP-LSE), we analyze its suboptimality and infeasibility with respect to (SDP). The dual of (SDP) is

$$\min_y \quad b^T y \quad \text{subject to} \quad \mathcal{A}^*(y) - C \succeq 0. \tag{DSDP}$$

We assume that (SDP) is feasible and a constraint qualification holds ensuring that strong duality is satisfied, and the primal and dual problems have finite optimal values. Let $(X^\star_{SDP}, y^\star_{SDP})$ be a primal-dual optimal pair. The optimality and infeasibility bounds that we obtain depend on the properties of the optimal dual solution, $y^\star_{SDP}$.

Lemma 3.2. *Let $(X^\star_{SDP}, y^\star_{SDP})$ be a primal-dual optimal solution of (SDP)–(DSDP) and let $\widehat{X}_\epsilon$ be an ϵ-optimal solution of (SDP-LSE). If $\beta > \|y^\star_{SDP}\|_1$, then*

$$\langle C, X^\star_{SDP}\rangle - \epsilon \leq \langle C, \widehat{X}_\epsilon\rangle \leq \langle C, X^\star_{SDP}\rangle + \|y^\star_{SDP}\|_1 \frac{\beta\frac{\log(2d)}{M} + \epsilon}{\beta - \|y^\star_{SDP}\|_1}, \tag{3.4}$$

and

$$\|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty \leq \frac{\beta\frac{\log(2d)}{M} + \epsilon}{\beta - \|y^\star_{SDP}\|_1}. \tag{3.5}$$

Proof. The bounds (3.4) and (3.5) are derived in Appendix A. □

Remark 3.1. Lemma 3.2 shows that $\widehat{X}_\epsilon$ is a near-feasible point to (SDP) with bounded objective function value such that the infeasibility and optimality bounds depend on the dual solution, $\|y^\star_{SDP}\|_1$, and the parameters β and M. If the parameter values β and M are specifically chosen to be $\beta = 2\|y^\star_{SDP}\|_1$ and $M = 2\frac{\log(2d)}{\epsilon}$, then $\widehat{X}_\epsilon$ satisfies $\|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty \leq 2\epsilon$ and the objective function value is upper bounded by $\langle C, X^\star_{SDP}\rangle + 2\epsilon\|y^\star_{SDP}\|_1$.

Remark 3.2. In some cases, it is difficult to produce truly feasible points from the generated near-feasible points. While in other cases, such as (MaxCut-SDP), it is fairly straightforward to make small modifications to a generated near-feasible point to produce a feasible point with a similar objective value. Furthermore, we will see that for (MaxCut-SDP), it is possible to reduce $\|y^\star_{SDP}\|_1$ to $\langle C, X^\star_{SDP}\rangle$, eliminating the unknown optimal dual variable from our bounds.

4 Approximation Algorithm for MaxCut

We now apply the general framework from Section 3 to give an implementation of the Goemans-Williamson approximation algorithm for MaxCut that uses only $\mathcal{O}(n)$ working memory. Recall from Section 1.1 that the standard SDP relaxation of MaxCut, (MaxCut-SDP) is a special case of (SDP) with a symmetric diagonally dominant cost matrix C and the constraint set $\{X \in \mathbb{S}^n : X \succeq 0, \operatorname{diag}(X) = \mathbb{1}\}$. A key challenge in applying the results of Lemma 3.2 in this setting is that the suboptimality and infeasibility bounds obtained depend on the optimal solution of the dual SDP (DSDP). With no *a priori* knowledge of the dual solution, selecting the value of parameter β is difficult and the additive error term in (3.4) could be quite high. Furthermore, setting the value of β to an arbitrarily large value increases the curvature constant.

For (MaxCut-SDP), we show how to apply Lemma 3.2 to obtain relative error bounds on the suboptimality and infeasibility of the output of Algorithm 2 without *a priori* knowledge of the dual optimal solution. This allows us to appropriately choose the parameters β and M in the penalized formulation (MaxCut-LSE). Moreover, it is possible to generate a feasible solution to MaxCut that nearly achieves the approximation guarantee of Goemans-Williamson method, by applying Algorithm 3 (see Section 4.2) to the Gaussian samples generated from the PSD matrix produced by Algorithm 2.

Let the constraint 'diag$(X) = \mathbb{1}$' in (MaxCut-SDP) be penalized with $\phi_M(\cdot)$ (3.1) and consider the modified problem

$$\max_X \ \langle C, X\rangle - \beta\phi_M(\mathrm{diag}(X) - \mathbb{1}) \quad \text{subject to} \quad \begin{cases} \mathrm{Tr}(X) \le n \\ X \succeq 0. \end{cases} \tag{MaxCut-LSE}$$

We will see that choosing $M = 4\frac{\log(2d)}{\epsilon}$ and $\beta = 4\mathrm{Tr}(C)$ in (MaxCut-LSE) allows us to derive a relative error bound on the objective function value of (MaxCut-SDP). The main result of this section is given as follows:

Theorem 4.1. *Let* (MaxCut-LSE) *be solved to* $\epsilon\mathrm{Tr}(C)$*-optimality using Algorithm 2 with* $\epsilon \in \left(0, \frac{1}{3}\right)$, $\eta = 1$, $p = \frac{\epsilon}{T(n,\epsilon)}$, *where* $T(n,\epsilon) = 64\frac{\log(2n)n^2}{\epsilon^2}$, *followed by the rounding scheme of Algorithm 3. For a diagonally dominant matrix* C*, this procedure generates a binary vector* w *that satisfies*

$$\alpha_{GW}(1 - 3\epsilon)\mathrm{opt} \le \mathbb{E}[w^T C w] \le \mathrm{opt}, \tag{4.1}$$

where opt *is the maximum of* $w^T C w$ *over the set* $w \in \{-1, 1\}^n$*. Algorithm 2 terminates after at most* $T(n,\epsilon)$ *iterations, where each iteration performs at most* $280\frac{n}{\epsilon}\log\left(\frac{2n}{\epsilon}\right)$ *matrix-vector multiplications. The working memory required is at most* $3n$ *numbers.*

4.1 Relative Error Bounds on Suboptimality and Infeasibility for (MaxCut-SDP)

In this subsection, we derive relative error bounds on the suboptimality and infeasibility for (MaxCut-SDP) at $\widehat{X}_\epsilon$, the output of Algorithm 2, when solving (MaxCut-LSE). This is an application of Lemma 3.2 with the key difference that any dependence on the dual optimal solution has been eliminated.

The dual of SDP relaxation of MaxCut is,

$$\min_y \ \sum_{i=1}^n y_i \quad \text{subject to} \quad \mathrm{diag}^*(y) - C \succeq 0. \tag{4.2}$$

Let $y^\star_{SDP}$ be an optimal solution of Problem (4.2). We first show that $\|y^\star_{SDP}\|_1$ is upper bounded by $2\mathrm{Tr}(C)$.

Lemma 4.1. *Let* $(X^\star_{SDP}, y^\star_{SDP})$ *be a primal-dual optimal pair for* (MaxCut-SDP) *and its dual* (4.2)*. If* C *is diagonally dominant, then*

$$\mathrm{Tr}(C) \le \langle C, X^\star_{SDP}\rangle = \|y^\star_{SDP}\|_1 \le 2\mathrm{Tr}(C). \tag{4.3}$$

Proof. For a symmetric diagonally dominant cost matrix C with nonnegative entries on the diagonal, it follows from the Gershgorin cirle theorem, that C must be a PSD matrix. Since C is PSD, $\mathrm{diag}^*(y) - C \succeq 0$ implies $y \ge 0$. Thus, the objective function of Problem (4.2) can be written as $\|y\|_1$. Moreover, the SDP relaxation of MaxCut satisfies Slater's condition, so

$$\langle C, X^\star_{SDP}\rangle = \|y^\star_{SDP}\|_1,$$

for a primal-dual optimal pair $(X^\star_{SDP}, y^\star_{SDP})$. To see that $\mathrm{Tr}(C) = \langle C, I\rangle \le \langle C, X^\star_{SDP}\rangle$, we simply note that identity matrix I is feasible for (MaxCut-SDP).

To prove $\langle C, X^\star_{SDP}\rangle \le 2\mathrm{Tr}(C)$, we use the fact that C is a diagonally dominant matrix with nonnegative entries on the diagonal. As such the matrix $2\mathrm{diag}^*(\mathrm{diag}(C)) - C$ is also symmetric diagonally dominant and has nonnegative diagonal entries. It follows that

$$2\mathrm{diag}^*(\mathrm{diag}(C)) - C \succeq 0. \tag{4.4}$$

Then,

$$\langle C, X^\star_{SDP}\rangle \le \langle 2\mathrm{diag}^*(\mathrm{diag}(C)), X^\star_{SDP}\rangle = 2\mathrm{Tr}(C), \tag{4.5}$$

where we used the fact that $X^\star_{SDP} \succeq 0$ and $\mathrm{diag}(X^\star_{SDP}) = \mathbb{1}$. □

Remark 4.1. If $C \succeq 0$, but not diagonally dominant, we have $\|y^\star_{SDP}\|_1 \leq n\lambda_{\max}(C) \leq n\mathrm{Tr}(C)$, since $y = \mathbb{1}\cdot\lambda_{\max}(C)$ is feasible for the dual problem (4.2). So, $\mathrm{Tr}(C) \leq \langle C, X^\star_{SDP}\rangle = \|y^\star_{SDP}\|_1 \leq n\mathrm{Tr}(C)$ in that case.

4.1.1 Optimality and feasibility bounds for (MaxCut-SDP)

By finding a near-optimal point for (MaxCut-LSE), the penalized relaxation of (MaxCut-SDP), we can obtain a near-feasible solution to (MaxCut-SDP) that has relative error ϵ. Note that a relative error bound is exactly what we need to obtain a multiplicative approximation guarantee for MaxCut.

Lemma 4.2. *Let $X^\star_{SDP}$ be an optimal solution of (MaxCut-SDP) and $\widehat{X}_\epsilon$ be an $\epsilon\mathrm{Tr}(C)$-optimal solution to (MaxCut-LSE) with $M = 4\frac{\log(2d)}{\epsilon}$ and $\beta = 4\mathrm{Tr}(C)$. Then*

$$\langle C, X^\star_{SDP}\rangle(1-\epsilon) \leq \langle C, \widehat{X}_\epsilon\rangle \leq \langle C, X^\star_{SDP}\rangle(1+\epsilon), \tag{4.6}$$

and

$$\|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty \leq \epsilon. \tag{4.7}$$

Proof. This result is an application of Lemma 3.2 for specific parameter values. Substituting the values of M and β, and using the inequality $\|y^\star_{SDP}\|_1 \leq 2\mathrm{Tr}(C)$ from Lemma 4.1, we see that

$$\|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty \leq \frac{\beta\frac{\log(2d)}{M} + \epsilon\mathrm{Tr}(C)}{\beta - \|y^\star_{SDP}\|_1} \leq \frac{2\epsilon\mathrm{Tr}(C)}{4\mathrm{Tr}(C) - 2\mathrm{Tr}(C)} = \epsilon.$$

Furthermore, combining the result $\langle C, X^\star_{SDP}\rangle = \|y^\star_{SDP}\|_1$ (from Lemma 4.1) with Lemma 3.2 gives the upper bound on $\langle C, \widehat{X}_\epsilon\rangle$. Finally, using the fact that $\langle C, X^\star_{SDP}\rangle \geq \mathrm{Tr}(C)$ and substituting in (3.4), gives

$$\langle C, \widehat{X}_\epsilon\rangle \geq \langle C, X^\star_{SDP}\rangle - \epsilon\mathrm{Tr}(C) \geq \langle C, X^\star_{SDP}\rangle(1-\epsilon). \qquad \square$$

Remark 4.2. If $C \succeq 0$ (not necessarily diagonally dominant), then the bounds in Lemma 4.2 hold for $M = (n+2)\frac{\log(2d)}{\epsilon}$ and $\beta = (n+2)\mathrm{Tr}(C)$.

4.2 Generating a Feasible Solution to MaxCut

We now show how to adapt the rounding procedure of Goemans-Williamson to our setting (Algorithm 3). The reason we need to modify the Goemans-Williamson scheme is because the zero-mean Gaussian random vector returned by Algorithm 2 has covariance that is not feasible for (MaxCut-SDP).

Algorithm 3: Generate a binary vector from a Gaussian vector

Input : A sample $\widehat{z}_\epsilon \sim \mathcal{N}(0, \widehat{X}_\epsilon)$ and $\mathrm{diag}(\widehat{X}_\epsilon)$
Output: A feasible solution to MaxCut $w = \mathrm{sign}(\overline{w})$

Function `GenerateSample`**:**
  Generate $\zeta \sim \mathcal{N}\left(0, I - \mathrm{diag}^*\left(\frac{\mathrm{diag}(\widehat{X}_\epsilon)}{\max(\mathrm{diag}(\widehat{X}_\epsilon))}\right)\right)$
  Set $\overline{w} = \frac{\widehat{z}_\epsilon}{\sqrt{\max(\mathrm{diag}(\widehat{X}_\epsilon))}} + \zeta$
return $\mathrm{sign}(\overline{w})$

Algorithm 3 can be used to generate a feasible solution of MaxCut from any PSD (covariance) matrix X. The first step of the algorithm generates n independent zero-mean random

variables with covariance defined by the diagonal entries of $I - \text{diag}^*\left(\frac{\text{diag}(\widehat{X}_\epsilon)}{\max(\text{diag}(\widehat{X}_\epsilon))}\right)$. The random vector $\overline{w}$ in step 3 is a sum of two independent zero-mean Gaussian random vectors. The covariance of this random vector $\overline{w}$ can be stated as

$$\overline{X} = \frac{\widehat{X}_\epsilon}{\max(\text{diag}(\widehat{X}_\epsilon))} + \left(I - \text{diag}^*\left(\frac{\text{diag}(\widehat{X}_\epsilon)}{\max(\text{diag}(\widehat{X}_\epsilon))}\right)\right) \tag{4.8}$$

so that $\overline{w} \sim \mathcal{N}(0, \overline{X})$. The matrix $\overline{X}$ is a sum of two PSD matrices and so is PSD. Moreover, $\text{diag}(\overline{X}) = \mathbb{1}$, so $\overline{X}$ is feasible for (MaxCut-SDP). We can then apply the standard analysis of the Goemans-Williamson rounding scheme to $\overline{X}$.

Goemans-Williamson rounding. For a PSD matrix C and a Gaussian random vector $\overline{w} \sim \mathcal{N}(0, \overline{X})$, such that $\text{diag}(\overline{X}) = \mathbb{1}$, Nesterov [26] derived a $\frac{2}{\pi}$-approximation bound,

$$\mathbb{E}_G[w^T C w] \geq \alpha \langle C, \overline{X} \rangle, \tag{4.9}$$

where $w = \text{sign}(\overline{w})$, $\alpha = \frac{2}{\pi}$ and $\mathbb{E}_G[\cdot]$ denotes the expectation over Gaussian random vectors. Moreover, if C is diagonally dominant, Goemans and Williamson [14] provide a tighter bound with $\alpha = \alpha_{GW} \approx 0.878$.

When the input of Algorithm 3 is an approximate solution of (MaxCut-SDP), we analyze the expected objective value of $\text{sign}(\overline{w})$.

Lemma 4.3. *Let $\epsilon \in \left(0, \frac{1}{2}\right)$, C be a diagonally dominant matrix, and let $\widehat{X}_\epsilon \succeq 0$ satisfy the bounds given in Lemma 4.2. If a binary vector $w = \text{sign}(\overline{w})$ is generated by Algorithm 3 with input $\widehat{z}_\epsilon \sim \mathcal{N}(0, \widehat{X}_\epsilon)$, then the expected value of $w^T C w$ satisfies*

$$\alpha_{GW}(1 - 2\epsilon)\text{opt} \leq \mathbb{E}_G[w^T C w] \leq \text{opt} \leq \langle C, X^\star_{SDP} \rangle, \tag{4.10}$$

where opt *is the optimal value of $w^T C w$ over the set $w \in \{\pm 1\}^n$.*

Proof. The objective function value of (MaxCut-SDP) at $\overline{X}$ is

$$\begin{aligned}
\langle C, \overline{X} \rangle &= \left\langle C, \frac{\widehat{X}_\epsilon}{\max(\text{diag}(\widehat{X}_\epsilon))} + \left(I - \text{diag}^*\left(\frac{\text{diag}(\widehat{X}_\epsilon)}{\max(\text{diag}(\widehat{X}_\epsilon))}\right)\right)\right\rangle && (4.11)\\
&\geq \frac{\langle C, \widehat{X}_\epsilon \rangle}{\max(\text{diag}(\widehat{X}_\epsilon))} && (4.12)\\
&\geq \frac{1-\epsilon}{1+\epsilon}\langle C, X^\star_{SDP} \rangle && (4.13)\\
&\geq (1 - 2\epsilon)\langle C, X^\star_{SDP} \rangle, && (4.14)
\end{aligned}$$

where (4.12) follows from the fact that both C and $I - \text{diag}^*\left(\frac{\text{diag}(\widehat{X}_\epsilon)}{\max(\text{diag}(\widehat{X}_\epsilon))}\right)$ are PSD and their inner product is greater than 0, (4.13) follows from Lemma 4.2, and (4.14) uses the fact that $\frac{1}{1+\epsilon} \geq 1 - \epsilon$ and $(1-\epsilon)^2 \geq 1 - 2\epsilon$. Substituting (4.14) in (4.9) gives the desired result. □

Note that the result in Lemma 4.3 holds irrespective of the algorithm used to compute $\widehat{X}_\epsilon$. We are now in a position to prove Theorem 4.1.

Proof of Theorem 4.1. Since we use Algorithm 2 to solve (MaxCut-LSE) with $p = \frac{\epsilon}{T(n,\epsilon)}$, the bounds in Lemma 4.2 are satisfied with probability at least $1 - \epsilon$. Thus, the bound $\langle C, \overline{X} \rangle \geq (1-2\epsilon)\langle C, X^\star_{SDP} \rangle$ also holds with probability at least $1-\epsilon$. Moreover, for any $\overline{X} \succeq 0$, $\langle C, \overline{X} \rangle \geq 0$,

and thus, a lower bound on the expected value of $\langle C, \overline{X}\rangle$ over the random initialization of the power method, i.e., $\mathbb{E}_P[\langle C, \overline{X}\rangle]$, at each iteration is,

$$\begin{aligned}\mathbb{E}_P[\langle C, \overline{X}\rangle] &\geq (1-2\epsilon)\langle C, X^\star_{SDP}\rangle(1-\epsilon)\\ &\geq (1-3\epsilon)\langle C, X^\star_{SDP}\rangle.\end{aligned}$$

The lower bound in (4.1) then follows from (4.9) because

$$\begin{aligned}\mathbb{E}[w^T C w] &= \mathbb{E}_P[\mathbb{E}_G[w^T C w]]\\ &\geq \alpha_{GW}\mathbb{E}_P[\langle C, \overline{X}\rangle]\\ &\geq \alpha_{GW}(1-3\epsilon)\langle C, X^\star_{SDP}\rangle\\ &\geq \alpha_{GW}(1-3\epsilon)\text{opt}.\end{aligned}$$

Bound on T, number of iterations of Algorithm 2. An upper bound on the curvature constant of (MaxCut-LSE) is $C_g^u = 16\frac{\text{Tr}(C)\log(2n)n^2}{\epsilon}$ since $\omega = 1$ and $\alpha = n$. Algorithm 2 converges to an $\epsilon\text{Tr}(C)$-optimal solution after at most $T = \frac{2C_g^u(1+\eta)}{\epsilon\text{Tr}(C)} - 2 \leq 64\frac{\log(2n)n^2}{\epsilon^2}$ iterations with probability at least $1 - Tp$.

Bound on number of iterations of power method at each t. From Lemma 2.1, the number of matrix-vector multiplications performed at iteration t of Algorithm 2 is at most $\frac{\lambda\alpha}{\delta}\log\left(\frac{n}{p^2}\right)$ with $\delta = \frac{1}{2}\eta\gamma_t C_g^u \approx \frac{1}{8}\epsilon\text{Tr}(C)$ and

$$\begin{aligned}\lambda &= \max_i |\lambda_i(\nabla g(v_t))|\\ &= \max_i |\lambda_i(C - \beta D)|,\end{aligned}$$

where D is a diagonal matrix with $d_{ii} \in [-1, 1]$. Thus, $\lambda \leq \lambda_{\max}(C) + \beta \leq 5\text{Tr}(C)$.

Substituting the value of p, and bounds on λ and δ in Lemma 2.1, the number of iterations performed by the power method and thus, the number of matrix-vector computations at each t is bounded by $280\frac{n}{\epsilon}\log\left(\frac{2n}{\epsilon}\right)$. Furthermore, at iteration t of Algorithm 2, we keep track of the sample z and $\mathcal{A}(X)$ which requires storage of $2n$ numbers. Furthermore, the working memory of the power method is n numbers. This leads to a total working memory of at most $3n$ numbers. □

Remark 4.3. If $C \succeq 0$, but not diagonally dominant, then the suboptimality bound (4.1) holds for $M = (n+2)\frac{\log(2d)}{\epsilon}$ and $\beta = (n+2)\text{Tr}(C)$. However, due to the dependence of the values of parameters β and M on n in this case, it takes $\mathcal{O}\left(\frac{\log(2n)n^4}{\epsilon^2}\right)$ iterations within Algorithm 2 to achieve the stated guarantee.

5 From Gaussian to Randomized Extreme-Point Sampling

Until now, we have focused on using Gaussian random vectors to represent PSD matrix decision variables in low memory, and showed how to modify the Frank-Wolfe algorithm to track these samples. In this section, we discuss a more flexible approach to sample-based representations of decision variables.

Consider the problem

$$\max_{x\in\mathcal{S}} \quad g(\mathcal{B}(x)), \tag{5.1}$$

where g is a smooth concave function, $\mathcal{B}$ is a linear map, and $\mathcal{S}$ is a compact, convex set.

If $\mathcal{S}$ is the set of trace-constrained PSD matrices, we can apply Algorithm 2 as seen in Section 3. When the decision variable is not a PSD matrix, it is not immediately clear whether

there is a natural analogue of the Gaussian sampling idea from Section 3.1. One way to proceed in this case, is to think of $\mathcal{S}$ as the set of expectations of random variables supported on the extreme points of $\mathcal{S}$. The analogue of the Gaussian sampling idea is to construct a Markov chain on the extreme points of the feasible region so that its expectation converges to an optimal solution of Problem (5.1). Note that the updates in the Frank-Wolfe algorithm at each iteration are generated as optimal solutions to linear optimization problem over a convex set and hence can be taken to be extreme points of the feasible region. This idea opens up the possibility of developing algorithms for solving Problem (5.1) that require low working memory by modifying Frank-Wolfe, as long as certain conditions on the feasible set $\mathcal{S}$ are satisfied.

Randomized extreme-point sampling. The basic idea of randomized extreme-point sampling of (5.2) is to modify the Frank-Wolfe algorithm (Algorithm 1) so its state is a random extreme point z_t with expectation x_t. To do this, at iteration t, we update the random extreme point via

$$z_{t+1} = \begin{cases} z_t & \text{with probability } 1-\gamma_t \\ h_t & \text{with probability } \gamma_t, \end{cases} \tag{5.2}$$

where h_t is an update direction that is an extreme point of $\mathcal{S}$. Note that this update direction is computed as in Algorithm 1 and is deterministic since it depends on the variable $v_t = \mathcal{B}(x_t)$ that we track along with the sample z_t.

The expected value of z_{t+1} is $\mathbb{E}[z_{t+1}] = (1-\gamma_t)\mathbb{E}[z_t] + \gamma_t h_t$. By induction, it follows that at every iteration t, $\mathbb{E}[z_t] = x_t$ and $\mathbb{E}[z_{t+1}] = x_{t+1} = (1-\gamma_t)x_t + \gamma_t h_t$. Since this is equivalent to the update rule of Algorithm 1 in expectation, the convergence rate given in Theorem 2.1 also holds for $\mathbb{E}[z_t]$. Thus, replacing the solution x_t at iterate t with a random sample z_t, we get Frank-Wolfe with randomized extreme-point sampling, the outline of which is given in Algorithm 4.

Algorithm 4: Outline of Frank-Wolfe Algorithm with Randomized Extreme-Point Sampling

Input : Problem (5.1)
Output: A sample z such that $\mathbb{E}[z] = \widehat{x}_\epsilon$, where $\widehat{x}_\epsilon$ is an ϵ-optimal solution of (5.1)

Function `FWRandom`:
  Initialize $x_0 \in \mathcal{S}$, $v_0 = \mathcal{B}(x_0)$ and set z_0 to be a random extreme point with $\mathbb{E}[z_0] = X_0$
  Set $t = 0$, $\gamma_t = \frac{2}{t+2}$
  while *stopping criteria is not satisfied* **do**
    Using `LMO`, compute the update direction h_t, and $q_t = \mathcal{B}(h_t)$
    Update z_t using (5.2)
    Set $v_{t+1} \leftarrow (1-\gamma)v_t + \gamma q_t$
    $t \leftarrow t+1$, $\gamma_t \leftarrow \frac{2}{t+2}$
  end
return z_t

In order to implement Algorithm 4 in low memory, we require that the conditions LowMemoryComputations and LowMemoryExtremePoints from Section 2 should be satisfied. These conditions state that it must be possible to carry out the computations required to compute the update direction in low memory, and that we should be able to represent the extreme points of the feasible set $\mathcal{S}$ in low memory. In the rest of the section, we look at the application of randomized extreme-point sampling to example problems which satisfy conditions LowMemoryComputations and LowMemoryExtremePoints.

5.1 Randomized Extreme-Point Sampling for SDPs with Rank-1 Extreme Points

When the feasible region consists of trace constrained PSD matrices, the randomized extreme-point sampling of (5.2) can be applied to Problem (5.1). A key feature of this constraint set is that all of its extreme points have rank zero or one. Here, the condition LowMemoryExtremePoints from Section 2 is met and the extreme points require much less memory than the size of the decision variable.

The additional flexibility of randomized extreme-point sampling means that this technique is also applicable to a larger class of spectrahedra (i.e., feasible regions of semidefinite programs). In seeking feasible regions for which the extreme rays have a low memory representation, it is natural to consider spectrahedra that have rank-1 extreme points. These have been classified fully by Blekherman, Sinn, and Velasco [4]. Rather than discuss this class in general, we focus on the particular case of PSD matrices that are sparse with respect to a chordal graph.

Given a graph $G = (V, E)$, and a $|V| \times |V|$ symmetric matrix X, we say that X *is sparse with respect to* G if $X_{ij} = 0$ whenever $(i, j) \notin E$ and $i \neq j$. Consider the convex set

$$\mathcal{S}_G = \{X \in \mathbb{S}^{|V|} \ : \ \mathrm{Tr}(X) \leq \alpha, \ X \succeq 0, \ \ X \text{ is sparse with respect to } G\}.$$

A graph G is said to be *chordal* if every cycle of G of length at least four has a chord. If G is chordal, then the extreme points of $\mathcal{S}_G$ have the following characterization.

Theorem 5.1. *If G is a chordal graph, then $X \in \mathcal{S}_G$ is an extreme point of $\mathcal{S}_G$ if and only if $X = 0$ or $X = uu^T$, where $u \in \mathbb{R}^n$, $\|u\|_2^2 = \alpha$ and the indices of nonzero entries in u form a clique of G.*

Sketch of proof. ($\Leftarrow$) For any graph G, this follows from the fact that any rank-1 element of $\mathcal{S}_G$ must be an extreme point.

($\Rightarrow$) This is a consequence of [1, Theorem 2.3], which states that any PSD matrix sparse with respect to a chordal graph decomposes as a sum of PSD matrices, each sparse with respect to some maximal clique of G. □

Thus, given an input chordal graph G and its set of maximal cliques, the extreme points of $\mathcal{S}_G$ have rank at most one, with the number of nonzero elements in the rank-1 factorization upper bounded by the size of the largest clique. The memory required to represent each extreme point is then bounded above by the number of vertices in the graph and satisfies the condition LowMemoryExtremePoints.

For Algorithm 4 to be a low memory algorithm, we also need to check that the condition LowMemoryComputations holds. Computing the update direction now requires solving one eigenvalue problem for each maximal clique of G. These can be solved serially each via the power method so that we get an update direction that is a zero-padded vector representing an extreme point of $\mathcal{S}_G$. The overall memory required by Algorithm 4 for these computations is still bounded by $\mathcal{O}(d + |V|)$, i.e., the dimension of the codomain of the linear map $\mathcal{B}$ and the size of the largest maximal clique in the chordal graph G. As such, the condition LowMemoryComputations is satisfied.

The difference between the feasible region of (BoundedSDP) and the feasible region $\mathcal{S}_G$ is the additional $\binom{|V|}{2} - |E|$ linear constraints. In Section 3.2, we dealt with such additional constraints by incorporating an associated penalty into the objective function at the expense of infeasibility and increasing the curvature constant of the objective. However, when G is chordal, the extreme points of $\mathcal{S}_G$ have a concise representation given by Theorem 5.1, and satisfy conditions LowMemoryComputations and LowMemoryExtremePoints. Using Algorithm 4 eliminates the need to penalize these additional constraints.

If the graph G is not chordal, we can combine the penalization approach and the chordal graph approach as follows. We add extra edges to the graph G to make a chordal graph $\overline{G}$

(known as a chordal cover). Because we have added extra edges to G to get $\overline{G}$, $\mathcal{S}_G \subseteq \mathcal{S}_{\overline{G}}$. The constraints corresponding to $E(\overline{G})\backslash E(G)$ could then be penalized in the objective function using the penalty function defined by equation (3.1). This gives a problem with a modified objective function and a convex feasible region where the decision variable is sparse with respect to the chordal graph $\overline{G}$. Using Algorithm 4 will now generate a near-feasible, near-optimal solution to the problem defined on the graph G.

5.2 Illustrating Randomized Extreme-Point Sampling

The applications considered until now have a PSD matrix decision variable. In this subsection, we illustrate the randomized extreme-point strategy for two problems with feasible regions that are not subsets of the PSD cone. We discuss the sensor selection problem of Joshi and Boyd [18] and the compressive sensing problem. We discuss assumptions on the problem data under which randomized extreme-point sampling can be implemented in low memory.

5.2.1 Sensor Selection Problem

In the sensor selection problem, we are given n noisy sensors, and the aim is to choose a subset of size k that allows us to estimate an unknown quantity with least uncertainty. If we consider linear sensors with additive Gaussian noise, the i-th sensor takes a measurement of the form $y_i = a_i^T\zeta + \xi_i$, where $\xi_i \sim \mathcal{N}(0,1)$ and $\zeta \in \mathbb{R}^m$. The confidence ellipsoid of the estimate obtained using a subset $I \subseteq \{1,\ldots,n\}$ of sensors has volume proportional to $\log\det\left(\left(\sum_{i\in I} x_i a_i a_i^T\right)^{-1}\right)$. The problem of choosing the subset of sensors of size k that minimizes the volume of this ellipsoid can be written as the following mixed integer convex program:

$$\max_x \quad \log\det\left(\sum_{i=1}^n x_i a_i a_i^T\right) \quad \text{subject to} \quad \begin{cases} \sum_{i=1}^n x_i = k \\ x_i \in \{0,1\}, \quad i = 1,\ldots,n, \end{cases} \tag{5.3}$$

where the variable x_i indicates the selection of sensor i for observation. By relaxing the integrality condition in Problem (5.3), we obtain the following standard convex relaxation studied by Joshi and Boyd [18],

$$\max_{x\in\mathcal{S}} \quad g(\mathcal{B}(x)) = \log\det\left(\mathcal{B}(x)\right) \tag{5.4}$$

where $\mathcal{B}(x) = \sum_{i=1}^n x_i a_i a_i^T$ and $\mathcal{S} = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i = k, 0 \le x_i \le 1, i = 1,\ldots,n\}$. The extreme points of $\mathcal{S}$ are vectors in $\mathbb{R}^n$ with exactly k nonzero entries each with value equal to one. These can be represented in low memory by storing the k indices of nonzero entries, which require $k\log(n)$ bits. As such, the condition LowMemoryExtremePoints is satisfied.

Now we look at the way we can find a feasible solution to Problem (5.3) in low working memory using randomized extreme-point sampling (Algorithm 4). This technique returns a random subset of sensors of size k, and so naturally integrates solving the convex relaxation with a method for rounding a feasible solution to Problem (5.4) to a feasible solution to Problem (5.3).

The objective function of Problem (5.4) is a smooth, convex function as long as $\sum_{i\in I} a_i a_i^T \succ 0$, whenever $|I| = k$. At iterate t,

$$\nabla g(\mathcal{B}(x^{(t)})) = [a_i^T A(x^{(t)}) a_i]_{i=1}^n, \tag{5.5}$$

where $A(x^{(t)}) = \left[\sum_{i=1}^n x_i^{(t)} a_i a_i^T\right]^{-1}$. The update direction is

$$h^{(t)} = \arg\max_{d\in\mathcal{S}} \langle \nabla g(\mathcal{B}(x^{(t)})), d\rangle. \tag{5.6}$$

which is equivalent to computing the k largest entries of the nonnegative vector $\nabla g(\mathcal{B}(x^{(t)}))$. For the LowMemoryComputations property to hold, we make two assumptions, (1) that $n \gg m^2$

and (2) that the sensor parameters a_i do not need to be stored explicitly. Then we can compute the update direction by generating each element of $\nabla g(\mathcal{B}(x^{(t)}))$ serially and only storing k largest elements and their indices. Thus, solving this subproblem requires us to store $\tilde{\mathcal{O}}(m^2+k)$ numbers (suppressing the $\log(n)$ memory required to represent a number between 1 and n), to store $A(x^{(t)})$ and k largest elements of $\nabla g(\mathcal{B}(x^{(t)}))$ and their indices.

In summary, by applying Algorithm 4, we can select k sensors that form a near-optimal solution for the convex relaxation (Problem (5.4)) in expectation with working memory of $\tilde{\mathcal{O}}(m^2+k)$ numbers (again suppressing logarithmic factors in n).

5.2.2 Compressive Sensing

Compressive sensing is used to reconstruct a sparse signal $x \in \mathbb{R}^n$ from a set of noisy linear measurements $y = Ax + w \in \mathbb{R}^m$ with $m \ll n$ [28]. If, in addition, x is nonnegative, a standard convex formulation of the problem is

$$\min_x \quad \frac{1}{2}\|Ax-y\|_2^2 \quad \text{subject to} \quad \begin{cases} \|x\|_1 \le \alpha \\ x \ge 0. \end{cases} \tag{5.7}$$

The extreme points of the feasible region $\mathcal{S} = \{x \in \mathbb{R}^n : \|x\|_1 \le \alpha, x \ge 0\}$ of Problem (5.7) are the origin and vectors that have a single nonzero element with value equal to α. An extreme point can now be represented as a singleton containing the index of the nonzero element, requiring $\log(n)$ bits of storage.

Let $\mathcal{B}(\cdot) : \mathbb{R}^n \to \mathbb{R}^m$ be defined as $\mathcal{B}(x) = Ax$, so that the objective function of Problem (5.7) is of the form, $g(\mathcal{B}(x)) = \frac{1}{2}\|\mathcal{B}(x) - y\|_2^2$. The update direction is computed as

$$h_t = \operatorname*{arg\,max}_{\substack{\|d\|_1 \le \alpha, \\ d \ge 0}} \langle \nabla g(\mathcal{B}(x_t)), d \rangle. \tag{5.8}$$

If $i^\star \in \arg\max_i \quad \nabla g(\mathcal{B}(x_t))_i$, then an optimal solution of Problem (5.8) is a vector with a single nonzero element indexed by $i^\star$ and whose value is equal to α (or the zero vector if $\nabla g(\mathcal{B}(x_t)) \le 0$). Since computing $i^\star$ is equivalent to finding the largest element in $\nabla g(\mathcal{B}(x_t))$, the LowMemoryComputations property will hold as long as we can generate the columns of A serially without explicitly storing them. This would be possible, for instance, if A were a partial Fourier matrix. The working memory of Algorithm 4 is effectively restricted to $\mathcal{O}(m)$ numbers required to store v and an extreme point.

Recovering a signal with k nonzero elements. The randomized extreme-point sampling algorithm returns a random index $i \in \{1, \ldots, n\}$. The index is a sample from the distribution defined by normalizing a near-optimal point for Problem (5.7). In Section 6, we briefly discuss how to recover a k-sparse approximation of the optimal solution to Problem (5.7). The main idea is to run the randomized extreme-point sampling algorithm multiple times and use a streaming algorithm to find the k most frequently occurring indices in the resulting stream.

6 Post-Processing of Samples

In the previous sections, we developed algorithms that generate samples of decision variables and their output is a collection of samples of an ϵ-optimal solution of a convex optimization problem of the form (5.1). When Gaussian sampling is used with Algorithm 2, the output zero-mean Gaussian samples have *covariance* that represents an ϵ-optimal solution of the input problem. Whereas, when randomized extreme-point sampling (Algorithm 4) is used, the output is a sample whose *expected value* represents an ϵ-optimal solution to the problem. In this section we

briefly discuss further processing that can done be on these samples to generate other memory-efficient approximations of that near-optimal solution. The general approach will be to make use of various streaming algorithms.

For instance, given a sequence of i.i.d. Gaussian samples $z_i \sim \mathcal{N}(0, X)$ generated by running Algorithm 2 for problems with a unique optimal solution, we would like to estimate either X itself or a *low memory approximation* of X. Low memory approximations of interest include rank-k approximations, Ω-sparse approximations or the top k principal components. When the samples are generated sequentially, processed and discarded, such covariance estimation or approximation problems can potentially be solved in a memory-efficient way using streaming algorithms. When the decision variable is (up to scaling) a probability vector, as in the compressing problem from Section 5, Algorithm 4 gives a stream of samples from that probability distribution. A natural post-processing task is to find a k-sparse approximation of the underlying decision variable. In the rest of the section, we briefly discuss how to post-process sampled solutions to construct such approximations of the solution of the underlying convex program.

Finding low-rank approximation of covariance matrix. Given a stream of i.i.d. samples $z_1, z_2, \ldots, z_N \sim \mathcal{N}(0, X)$, we can use the method proposed by Tropp et al. [30] to obtain a rank-r approximation of the sample covariance matrix $X_N = \frac{1}{N}\sum_{i=1}^{N} z_i z_i^T$. The method involves generating and updating a linear sketch $Y_N = X_N\Omega$ of the sample covariance matrix, where $\Omega \in \mathbb{R}^{n\times k}$ is a fixed matrix with i.i.d. standard Gaussian entries, and $r \leq k \leq n$. Given a new random sample $z_{N+1} \sim \mathcal{N}(0, X)$, the sketch is updated via $Y_{N+1} = \frac{N}{N+1}Y_N + \frac{1}{N+1}z(z^T\Omega)$. Note that this sketch requires $\Theta(kn)$ memory and the computational cost of updating the sketch is $\Theta(kn)$. Furthermore, using [30, Algorithm 3] it is possible to reconstruct a rank-r approximation $\widehat{X}_N$ of the sample covariance matrix X_N from Y_N. In particular, if $k \sim \Theta(r/\epsilon)$, it is possible to generate $\widehat{X}_N$ such that $\mathbb{E}\|X_N - \widehat{X}_N\|_1 \leq (1+\epsilon)\|X_N - [X_N]_r\|_1$, where $[X_N]_r$ is the best rank-r approximation of the sample covariance and $\|\cdot\|_1$ is the Schatten-1 norm. By choosing sufficiently many samples N, we can ensure that $\mathbb{E}\left[\|X - \widehat{X}_N\|_1\right] \leq (1+2\epsilon)\|X - [X]_r\|_1$, where X is the population covariance of the samples and $[X]_r$ is its best rank-r approximation. This means that using $\mathcal{O}(nr/\epsilon)$ memory, we can post process the sampled output of Algorithm 2 to obtain a rank-r approximation of a near-optimal solution of (BoundedSDP).

Finding top eigenspace using streaming PCA. Given i.i.d. samples $z_1, z_2, \ldots, z_N \sim \mathcal{N}(0, X)$ with bounded norm, we may also want to estimate the eigenspace corresponding to the k largest eigenvectors. Unlike estimating a rank-k approximation, we require a nonzero eigengap between the eigenvalues λ_k and λ_{k+1} of the covariance matrix for this problem to be well defined. This problem is a variant of streaming PCA which computes a k-dimensional subspace that best contains n-dimensional samples presented sequentially. The SPCA algorithm given by Oja and Karhunen [27] keeps track of an $n \times k$ matrix with orthonormal columns, say Q_N, at each step N. When a new sample $z_{N+1} \sim \mathcal{N}(0, X)$ with bounded norm is received, it performs the update $Q_{N+1} = \mathrm{QR}(Q_N + \gamma_N z_{N+1}(z_{N+1}^T Q_N))$ where $\mathrm{QR}(\cdot)$ computes the QR decomposition so that the matrix Q_{N+1} has orthonormal columns and $\gamma_N > 0$ is a step size parameter. Furthermore, from [23, Theorem 1], we see that using SPCA, it is possible to compute the top k eigenspace approximately with prespecified error by choosing sufficiently many samples. Since the samples are processed sequentially and discarded, the streaming PCA model requires no more than $\mathcal{O}(kn)$ memory to compute and store the top k eigenspace of the covariance matrix.

Finding a signal vector with k nonzero elements. The output of Algorithm 4, when applied to the compressive sensing problem, is a stream of data where each data point represents an index of a single nonzero element in the signal. As seen in Section 5.2.2, the goal of compressive sensing is to recover a signal with k nonzero elements. By recovering the frequency of k most

frequently occurring indices from this data stream, we recover a signal with k nonzero elements that satisfies the constraints (up to scaling) for the compressive sensing problem. The counter based technique proposed by Metwally, Agrawal, and El Abbadi [25] is a memory-efficient way to approximately compute these elements by only keeping track of the counts of occurrence of few elements, say $m \geq k$, at a time. When a new element z_i in the stream arrives, the count of the element is updated by one if it is being tracked at that time. Otherwise, it displaces the element with the lowest count in the list and the count of the new element is set to the count value of the element it displaces plus one. The data structure is a linked list such that the elements are stored in the decreasing order of their frequency. Furthermore, by setting $m = \frac{1}{\epsilon f_k}$, where f_k is the frequency of the k-th most common element, it is possible to find the top k frequently occurring elements such that the frequency of each element is at least $(1-\epsilon)f_k$ [25, Theorem 6].

7 Discussion

Comparison with Yurtsever et al. [36]. In Section 3, we saw that it is possible to generate and store samples of a near-feasible, near-optimal solution to (SDP) using Algorithm 2 with working memory that is independent of the approximation parameter ϵ and limited to $\mathcal{O}(d+n)$. This memory requirement differs from the $\mathcal{O}(d+rn/\epsilon)$ working memory required by the algorithm given by Yurtsever et al. [36]. The difference arises due the fact that we only aim to provide a sampled representation of the approximate solution rather than generate a near best rank r approximation of the solution, or recover the exact solution matrix. In the special case of MaxCut, our sample-based representation is sufficient to implement the Goemans-Williamson rounding scheme. This allows us to implement an $(1-\epsilon)\alpha_{GW}$ approximation algorithm for MaxCut using memory linear in n and independent of ϵ. Yurtsever et al. [36] have performed numerical experiments which show that their method is capable of handling (MaxCut-SDP) with $n \approx 8 \cdot 10^6$ on a computer with 16 GB RAM. Our preliminary numerical results (see Section 7.1) are perhaps less promising in terms of practical convergence rate.

Alternative algorithms with Gaussian sampling. The Frank-Wolfe algorithm is well suited for Gaussian and extreme-point sampling when the extreme points have low rank. It is interesting to consider which other algorithms can be modified to track a sampled representation of the decision variable rather than the decision variable itself. For example, MaxCut algorithm given by Klein and Lu [20] generates a rank-1 update at each iteration and its output is used to produce a factorization of an approximate solution to (MaxCut-SDP). The structure of the updates and the computations required in their algorithm are structurally similar enough to our approach that the matrix iterates in [20] can be systematically replaced with Gaussian samples.

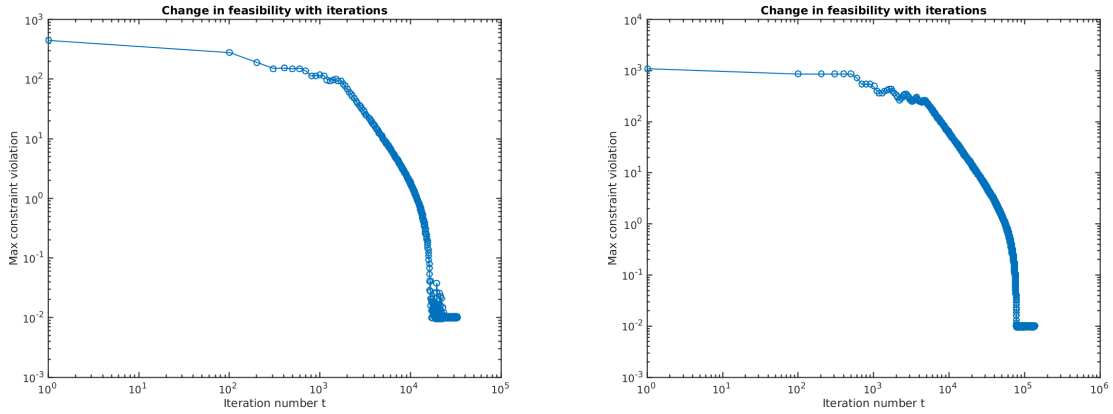
Another method where this could be done is the Matrix Multiplicative Weights (MMW) method, where the update to the variable takes the form $x_{t+1} \leftarrow x_t \exp(c(x))$ with $c(x)$ being a feedback function from the previous iterates. In the case of SDPs, this method requires computing the matrix exponential to generate the updates. Carmon et al. [8] provide an algorithm [8, Algorithm 1] for solving (SDP) using MMW method which relieves the computational burden of generating the matrix exponential. They do so by restricting the update to be a rank-1 sketch which is the result of multiplying the matrix exponential with a random vector drawn from a standard Gaussian distribution. This rank-1 sketch is computed using Lanczos algorithm without actually generating the matrix exponential. Again we expect that it should be possible to use the idea of randomized extreme-point sampling to get linear working memory while implementing this algorithm.

Using Gaussian sampling for other rounding schemes. There are a number of other approximation algorithms that involve solving (MaxCut-SDP) (with appropriate cost matrix C) and then rounding the Gaussian samples with covariance given by the SDP solution. Examples include MAX-2SAT [14] and the maximization of indefinite binary quadratic forms studied by Charikar and Wirth [9] and Megretski [24]. Our approach allows these approximation algorithms to also be solved in $\mathcal{O}(n)$ working memory. It would be interesting to investigate which other rounding schemes can be implemented in a memory-efficient way using modifications of our approach.

7.1 Preliminary Computational Results

We conclude with some preliminary computational experiments for MAXCUT. The algorithms we propose are simple to implement, and offer a scope for modification and improvement. Our aim here is to illustrate this simplicity and to identify possible areas for future algorithmic developments. We generated random graphs with sizes $n = 800$ to $n = 2000$ and with average degree of 10. Note that all the graphs had unweighted edges. The computations were performed using MATLAB R2018b on a machine with 8GB RAM and 4 cores. The input parameter values for MAXCUT were set as $d = n$, $\alpha = n$, $\omega = 1$, $\epsilon = 0.1$ $\beta = 4\mathrm{Tr}(C)$ and $M = 4\frac{\log(2d)}{\epsilon}$.

The key observation during the implementation was that the working memory at any time during the algorithm was restricted to at most $20n$, which is linear in the size of the graph as claimed. We tracked the change in infeasibility, which also determines an upper bound on sub-optimality of the solution, with each iteration. The plot of $\log(\|\mathcal{A}(X) - b\|_\infty)$ vs $\log(t)$ (iteration number) for $n = 800$ and $n = 2000$ is shown in Figure 1.



(a) $n = 800$ (b) $n = 2000$

Figure 1: Plot of $\log(\| \mathcal{A}(X_t) - b \|_\infty)$ vs $\log(t)$.

By comparing the two plots, we also noticed that the rate of the change in error observed was similar for problems with different sizes. For both problems, it was observed that during the initial phase, the rate of change in infeasibility is small. However, after a fixed percentage of total iterations, there is a steady decrease in the error and finally, it converges to a value about 10 times smaller than $\epsilon = 0.1$ in both cases.

Algorithm 2 was implemented exactly as given in Section 3 without any enhancements with the exception of using `eigs` command in MATLAB instead of the power method to compute eigenvectors in the `LMO` subroutine at each iteration. Note that the bounds in Theorem 4.1 are satisfied by the iterates. As the convergence is quite slow for first 10^3 iterations for $n = 800$, there is room for improvement in the design of the algorithm. We conjecture that this slow initial convergence is due to the approach taken to penalize the constraints and identify this as a natural direction for future algorithmic work. Currently, each iteration also requires computing

the leading eigenvector. This could be improved with warm start if we know the approximate subspace in which the eigenvector lies which might become clearer as the algorithm reaches the near-feasible, near-optimal solution to the input problem. These improvements can potentially lead to a more practical low memory method, based on the ideas presented in this paper.

References

[1] Jim Agler, William Helton, Scott McCullough, and Leiba Rodman. Positive semidefinite matrices with a given sparsity pattern. *Linear algebra and its applications*, 107:101–149, 1988.

[2] Alexander I. Barvinok. Problems of distance geometry and convex properties of quadratic maps. *Discrete & Computational Geometry*, 13(2):189–202, 1995.

[3] Srinadh Bhojanapalli, Anastasios Kyrillidis, and Sujay Sanghavi. Dropping convexity for faster semi-definite optimization. In *Conference on Learning Theory*, pages 530–582, 2016.

[4] Grigoriy Blekherman, Rainer Sinn, and Mauricio Velasco. Do sums of squares dream of free resolutions? *SIAM Journal on Applied Algebra and Geometry*, 1(1):175–199, 2017.

[5] Nicolas Boumal, Vlad Voroninski, and Afonso Bandeira. The non-convex burer-monteiro approach works on smooth semidefinite programs. In *Advances in Neural Information Processing Systems*, pages 2757–2765, 2016.

[6] Nicolas Boumal, Pierre-Antoine Absil, and Coralia Cartis. Global rates of convergence for nonconvex optimization on manifolds. *IMA Journal of Numerical Analysis*, 39(1):1–33, 2018.

[7] Samuel Burer and Renato DC Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357, 2003.

[8] Yair Carmon, John C Duchi, Sidford Aaron, and Tian Kevin. A rank-1 sketch for matrix multiplicative weights. In *Proceedings of the Thirty-Second Conference on Learning Theory*, volume 99, pages 589–623. PMLR, 2019.

[9] Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending Grothendieck's inequality. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 54–60. IEEE, 2004.

[10] Yudong Chen and Martin J Wainwright. Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees. *arXiv preprint arXiv:1509.03025*, 2015.

[11] Lijun Ding, Alp Yurtsever, Volkan Cevher, Joel A Tropp, and Madeleine Udell. An optimal-storage approach to semidefinite programming using approximate complementarity. *arXiv preprint arXiv:1902.03373*, 2019.

[12] Bassem Fares, Dominikus Noll, and Pierre Apkarian. Robust control via sequential semidefinite programming. *SIAM Journal on Control and Optimization*, 40(6):1791–1820, 2002.

[13] Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[14] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[15] Michael D Grigoriadis and Leonid G Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4 (1):86–107, 1994.

[16] Elad Hazan. Sparse approximate solutions to semidefinite programs. In *Latin American symposium on theoretical informatics*, pages 306–316. Springer, 2008.

[17] Martin Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *ICML (1)*, pages 427–435, 2013.

[18] Siddharth Joshi and Stephen Boyd. Sensor selection via convex optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, 2008.

[19] Michel Journée, Francis Bach, P-A Absil, and Rodolphe Sepulchre. Low-rank optimization on the cone of positive semidefinite matrices. *SIAM Journal on Optimization*, 20(5):2327–2351, 2010.

[20] Philip Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 338–347, 1996.

[21] Philip N Klein and Hsueh-I Lu. Space-efficient approximation algorithms for MaxCut and Coloring semidefinite programs. In *International Symposium on Algorithms and Computation*, pages 388–398. Springer, 1998.

[22] Jacek Kuczyński and Henryk Woźniakowski. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. *SIAM journal on matrix analysis and applications*, 13(4):1094–1122, 1992.

[23] Chun-Liang Li, Hsuan-Tien Lin, and Chi-Jen Lu. Rivalry of two families of algorithms for memory-restricted streaming PCA. In *Artificial Intelligence and Statistics*, pages 473–481, 2016.

[24] Alexandre Megretski. Relaxations of quadratic programs in operator theory and system analysis. In *Systems, approximation, singular integral operators, and related topics*, pages 365–392. Springer, 2001.

[25] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, pages 398–412. Springer, 2005.

[26] Yu Nesterov. Semidefinite relaxation and nonconvex quadratic optimization. *Optimization methods and software*, 9(1-3):141–160, 1998.

[27] Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications*, 106(1):69–84, 1985.

[28] Irena Orović, Vladan Papić, Cornel Ioana, Xiumei Li, and Srdjan Stanković. Compressive sensing in signal processing: algorithms and transform domain formulations. *Mathematical Problems in Engineering*, 2016, 2016.

[29] Gábor Pataki. On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of operations research*, 23(2):339–358, 1998.

[30] Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Fixed-rank approximation of a positive-semidefinite matrix from streaming data. In *Advances in Neural Information Processing Systems*, pages 1225–1234, 2017.

[31] Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Practical sketching algorithms for low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1454–1485, 2017.

[32] Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM review*, 38(1): 49–95, 1996.

[33] Lieven Vandenberghe, V Ragu Balakrishnan, Ragnar Wallin, Anders Hansson, and Tae Roh. Interior-point algorithms for semidefinite programming problems derived from the KYP lemma. In *Positive polynomials in control*, pages 195–238. Springer, 2005.

[34] Irène Waldspurger and Alden Waters. Rank optimality for the Burer-Monteiro factorization. *arXiv preprint arXiv:1812.03046*, 2018.

[35] Alp Yurtsever, Madeleine Udell, Joel A Tropp, and Volkan Cevher. Sketchy decisions: Convex low-rank matrix optimization with optimal storage. *arXiv preprint arXiv:1702.06838*, 2017.

[36] Alp Yurtsever, Joel A Tropp, Olivier Fercoq, Madeleine Udell, and Volkan Cevher. Scalable semidefinite programming. *arXiv preprint arXiv:1912.02949*, 2019.

A Proof of Lemma 3.2

Proof. There are three inequalities to prove.

Lower bound on the objective function value, $\langle C, \widehat{X}_\epsilon \rangle$. Let $X^\star_{FW}$ be an optimal solution to (SDP-LSE). After the stopping criteria of Algorithm 2 is satisfied, the following holds:

$$g(\mathcal{B}(\widehat{X}_\epsilon)) \geq g(\mathcal{B}(X^\star_{FW})) - \epsilon \geq g(\mathcal{B}(X^\star_{SDP})) - \epsilon \tag{A.1}$$

since $X^\star_{SDP}$ is feasible for (SDP-LSE). Thus,

$$\langle C, \widehat{X}_\epsilon \rangle - \beta\phi_M(\mathcal{A}(\widehat{X}_\epsilon) - b) \geq \langle C, X^\star_{SDP} \rangle - \beta\phi_M(\mathcal{A}(X^\star_{SDP}) - b) - \epsilon. \tag{A.2}$$

The lower bound in (3.4) follows since $\phi_M(\mathcal{A}(X^\star_{SDP}) - b) \leq \phi_M(\mathcal{A}(\widehat{X}_\epsilon) - b)$.

Upper bound on the objective function value, $\langle C, \widehat{X}_\epsilon \rangle$. Let the Lagrangian of (SDP) be defined as

$$L(X, y) = \langle C, X \rangle - y^T(\mathcal{A}(X) - b).$$

For a primal-dual optimal pair, $(X^\star_{SDP}, y^\star_{SDP})$ and any $X \succeq 0$, the following holds,

$$L(X, y^\star_{SDP}) \leq L(X^\star_{SDP}, y^\star_{SDP}). \tag{A.3}$$

Since $\widehat{X}_\epsilon \succeq 0$, from (A.3), we can write

$$\begin{aligned} \langle C, \widehat{X}_\epsilon \rangle - y^{\star T}_{SDP}(\mathcal{A}(\widehat{X}_\epsilon) - b) &\leq \langle C, X^\star_{SDP} \rangle - y^{\star T}_{SDP}(\mathcal{A}(X^\star_{SDP}) - b) \\ &= \langle C, X^\star_{SDP} \rangle. \end{aligned} \tag{A.4}$$

The upper bound on $\langle C, \widehat{X}_\epsilon \rangle$ can be written as,

$$\langle C, \widehat{X}_\epsilon \rangle \leq \langle C, X^\star_{SDP} \rangle + y^{\star T}_{SDP}(\mathcal{A}(\widehat{X}_\epsilon) - b) \tag{A.5}$$

$$\leq \langle C, X^\star_{SDP} \rangle + \|y^\star_{SDP}\|_1 \|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty. \tag{A.6}$$

Bound on infeasibility, $\|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty$. We rewrite (A.1) as,

$$\begin{aligned}\beta\phi_M(\mathcal{A}(\widehat{X}_\epsilon) - b) &\leq \langle C, \widehat{X}_\epsilon \rangle - \langle C, X^\star_{SDP} \rangle + \beta\phi_M(\mathcal{A}(X^\star_{SDP}) - b) + \epsilon \\ &\leq \|y^\star_{SDP}\|_1 \|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty + \beta\phi_M(\mathcal{A}(X^\star_{SDP}) - b) + \epsilon \quad \text{(from (A.6))}.\end{aligned}$$

Now $\phi_M(\mathcal{A}(X^\star_{SDP}) - b) = \frac{\log(2d)}{M}$ and, from Proposition 3.2, we know that $\|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty \leq \phi_M(\mathcal{A}(\widehat{X}_\epsilon) - b)$. So,

$$\beta\|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty \leq \|y^\star_{SDP}\|_1 \|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty + \beta\frac{\log(2d)}{M} + \epsilon. \tag{A.7}$$

Since $\beta > \|y^\star_{SDP}\|_1$ by assumption,

$$\begin{aligned}&(\beta - \|y^\star_{SDP}\|_1)\, \|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty \leq \beta\frac{\log(2d)}{M} + \epsilon \\ &\Rightarrow \|\mathcal{A}(\widehat{X}_\epsilon) - b\|_\infty \leq \frac{\beta\frac{\log(2d)}{M} + \epsilon}{\beta - \|y^\star_{SDP}\|_1}.\end{aligned}$$

So, we get a bound on infeasibility that depends on $\|y^\star_{SDP}\|_1$, M and β.

Revisiting the upper bound on $\langle C, \widehat{X}_\epsilon \rangle$. Substituting the bound on infeasibility into (A.6) gives

$$\langle C, \widehat{X}_\epsilon \rangle \leq \langle C, X^\star_{SDP} \rangle + \|y^\star_{SDP}\|_1 \frac{\beta\frac{\log(2d)}{M} + \epsilon}{\beta - \|y^\star_{SDP}\|_1}.$$ □