Faster and Simpler Width-Independent Parallel Algorithms for Positive Semidefinite Programming

Richard Peng Kanat Tangwongsan
Carnegie Mellon University
{yangp, ktangwon}@cs.cmu.edu

ABSTRACT

This paper studies the problem of finding a $(1+\varepsilon)$-approximate solution to positive semidefinite programs. These are semidefinite programs in which all matrices in the constraints and objective are positive semidefinite and all scalars are non-negative. At *FOCS'11*, Jain and Yao gave an NC algorithm that requires $O(\frac{1}{\varepsilon^{13}}\log^{13} m \log n)$ iterations on input n constraint matrices of dimension m-by-m, where each iteration performs at least $\Omega(m^\omega)$ work since it involves computing the spectral decomposition.

We present a simpler NC parallel algorithm that on input with n constraint matrices, requires $O(\frac{1}{\varepsilon^4}\log^4 n \log(\frac{1}{\varepsilon}))$ iterations, each of which involves only simple matrix operations and computing the trace of the product of a matrix exponential and a positive semidefinite matrix. Further, given a positive SDP in a factorized form, the total work of our algorithm is nearly-linear in the number of non-zero entries in the factorization. Our algorithm can be viewed as a generalization of Young's algorithm and analysis techniques for positive linear programs (Young, *FOCS'01*) to the semidefinite programming setting.

Categories and Subject Descriptors: F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity

General Terms: Algorithms, Theory

Keywords: Parallel algorithms, semidefinite programming, covering semidefinite programs, approximation algorithms

1. INTRODUCTION

Semidefinite programming (SDP), alongside linear programming (LP), has been an important tool in approximation algorithms, optimization, and discrete mathematics. In the context of approximation algorithms alone, it has emerged as a key technique which underlies a number of impressive results that substantially improve the approximation ratios.

SPAA'12, June 25–27, 2012, Pittsburgh, Pennsylvania, USA.

In order to solve a semidefinite program, algorithms from the linear programming literature such as Ellipsoid or interior-point algorithms [GLS93] can be applied to derive near exact solutions. But this can be very costly. As a result, finding efficient approximations to such problems is a critical step in bringing these results closer to practice.

From a parallel algorithms standpoint, both LPs and SDPs are P-complete to even approximate to any constant accuracy, suggesting that it is unlikely that they have a polylogarithmic depth algorithm. For linear programs, however, the special case of positive linear programs, first studied by Luby and Nisan [LN93], has an algorithm that finds a $(1+\varepsilon)$-approximate solution in $O(\text{poly}(\frac{1}{\varepsilon}\log n))$ iterations. This weaker approximation guarantee is still sufficient for approximation algorithms (e.g., solutions to vertex cover and set cover via randomized rounding), spurring interest in studying these problems in both sequential and parallel contexts (see, e.g., [LN93, PST95, GK98, You01, KY07, KY09]).

The importance of problems such as MaxCut and Sparsest Cut has led to the identification and study of positive SDPs. The first definition of positive SDPs was due to Klein and Lu [KL96], who used it to characterize the MaxCut SDP. The MaxCut SDP can be viewed as a direct generalization of positive (packing) LPs. More recent work defines the notion of positive packing SDPs [IPS11], which captures problems such as MaxCut, sparse PCA, and coloring; and the notion of covering SDPs [IPS10], which captures the ARV relaxation of Sparsest Cut among others. The bulk of work in this area tends to focus on developing fast sequential algorithms for finding a $(1+\varepsilon)$-approximation, leading to a series of nice sequential algorithms (e.g., [AHK05, AK07, IPS11, IPS10]). The iteration count for these algorithms, however, depends on the so-called "width" parameter of the input program or some parameter of the spectrum of the input program. In some instances, the width parameter can be as large as $\Omega(n)$, making it a bottleneck in the depth of direct parallelization. Most recently, Jain and Yao [JY11] studied a particular class of positive SDPs and gave the first positive SDP algorithm whose work and depth are independent of the width parameter (commonly known as width-independent algorithms).

Our Work. We present a simple algorithm that offers the same approximation guarantee as [JY11] but has less work-depth complexity. Each iteration of our algorithm involves only simple matrix operations and computing the trace of the product of a matrix exponential and a positive semidefinite matrix. Furthermore, our proof only uses elementary

linear-algebraic techniques and the now-standard Golden-Thompson inequality.

The input consists of an accuracy parameter $\varepsilon > 0$ and a positive semidefinite program (PSDP) in the following standard primal form:

$$\begin{array}{lll} \text{Minimize} & \mathbf{C} \bullet \mathbf{Y} & \\ \text{Subject to:} & \mathbf{A}_i \bullet \mathbf{Y} \geq b_i & \text{for } i = 1, \ldots, n \\ & \mathbf{Y} \succcurlyeq \mathbf{0}, & \end{array} \tag{1.1}$$

where the matrices $\mathbf{C}, \mathbf{A}_1, \ldots, \mathbf{A}_n$ are m-by-m symmetric positive semidefinite matrices, $\bullet$ denotes the pointwise dot product between matrices (see Section 2), and the scalars $b_1, \ldots, b_n$ are non-negative reals. This is a subclass of SDPs where the matrices and scalars are "positive" in their respective settings. We also make the now-standard assumption that the SDP has strong duality. Our main result is as follows:

Theorem 1.1 (Main Theorem) *Given a primal positive SDP involving $m \times m$ matrices with n constraints and an accuracy parameter $\varepsilon > 0$, there is an algorithm* `approxPSDP` *that produces a $(1+\varepsilon)$-approximation in $O(\frac{1}{\varepsilon^4} \log^4 n \log(\frac{1}{\varepsilon}))$ iterations, where each iteration involves computing matrix sums and a special primitive that computes $\exp(\mathbf{\Phi}) \bullet \mathbf{A}$ in the case when $\mathbf{\Phi}$ and $\mathbf{A}$ are both positive semidefinite.*

The theorem quantifies the cost of our algorithm in terms of the number of iterations. The work and depth bounds implied by this theorem vary with the format of the input and how the matrix exponential is computed in each iteration. As we will discuss in Section 4, with input given in a suitable form, our algorithm runs in nearly-linear work and polylogarithmic depth. For comparison, the algorithm given in [JY11] requires $O(\frac{1}{\varepsilon^{13}} \log^{13} m \log n)$ iterations, each of which involves computing spectral decompositions using least $\Omega(m^\omega)$ work.

Recently, in an independent work, Jain and Yao [JY12] gave a similar algorithm for positive SDPs that is also based on Young's algorithm. Their algorithm solves a class of SDPs which contains both packing and diagonal covering constraints. Since matrix packing conditions between diagonal matrices are equivalent to point-wise conditions of the diagonal entries, these constraints are closer to a generalization of positive covering LP constraints. We believe that removing this restriction on diagonal packing matrices would greatly widen the class of problems included in this class of SDPs and discuss possibilities in this direction in Section 5.

1.1 Overview

All the parallel positive SDP algorithms to date can be seen as generalizations of previous works on positive linear programs (positive LPs). In the positive LPs literature, Luby and Nisan (LN) were the first to give a parallel algorithm for approximately solving a positive LP [LN93]. This algorithm provided the foundations for the algorithm given in [JY11], which like the LN algorithm, also works directly on the primal program. Using the dual as guide, the update step is intricate as their analysis is based on carefully analyzing the eigenspaces of a particular matrix before and after each update. Each of these iterations involves computing the spectral decomposition.

Our algorithm follows a different approach, based on the algorithm of Young [You01] for positive LPs. At the core of our algorithm is an algorithm for solving the decision version of the dual program. We derive this core routine by generalizing the algorithm and analysis techniques of Young [You01], using the matrix multiplicative weights update (MMWU) mechanism in place of Young's "soft" min and max. This leads to a simple algorithm: besides standard operations on (sparse) matrix, the only special primitive needed is the matrix dot product $\exp(\mathbf{\Phi}) \bullet \mathbf{A}$, where $\mathbf{\Phi}$ and $\mathbf{A}$ are both positive semidefinite.

More specifically, our algorithm works with normalized primal/dual programs shown in Figure 1.1. This is without loss of generality because any input program can be transformed into the normalized form by "dividing through" by $\mathbf{C}$ (see Appendix A). We solve a normalized SDP by resorting to an algorithm for its decision version and binary search. In particular, we design an algorithm with the property that given a goal value $\widehat{o}$, either find a dual solution $\boldsymbol{x} \in \mathbb{R}^+_n$ to (1.2-D) with objective at least $(1-\varepsilon)\widehat{o}$, or a primal solution $\mathbf{Y}$ to (1.2-P) with objective at most $\widehat{o}$. Furthermore, by scaling the $\mathbf{A}_i$'s, it suffices to only consider the case where $\widehat{o} = 1$. With this algorithm, the optimization version can be solved by binary searching on the objective a total of at most $O(\log(\frac{n}{\varepsilon}))$ iterations.

Intuitions. For intuition about packing SDPs and the matrix multiplicative weights update method for finding approximate solutions, a useful analogy of the decision problem is that of packing a (fractional) amount of ellipses into the unit ball. Figure 2 provides an example involving 3 matrices (ellipses) in 2 dimensions. Note that $\mathbf{A}_1$ and $\mathbf{A}_2$ are axis-aligned, so their sum is also an axis-aligned sum in this case. In fact, positive linear programs in the broader context corresponds exactly to the restriction of all ellipsoids being axis-aligned. In this setting, the algorithm of [You01] can be viewed as creating a penalty function by weighting the length of the axises using an exponential function. Then, ellipsoids with sufficiently small penalty subject to this function have their weights increased. However, once we allow general ellipsoids such as $\mathbf{A}_3$, the resulting sum will no longer be axis-aligned. In this setting, a natural extension is to take the exponential of the semi-major axises of the resulting ellipsoid instead. As we will see in later sections, the rest of Young's technique for achieving width-independence can also be adapted to this more general setting.

Work and Depth. We now discuss the work and depth bounds of our algorithm. The main cost of each iteration of our algorithm comes from computing the dot product between a matrix exponential and a PSD matrix. Like in the sequential setting [AHK05, AK07], we need to compute for each iteration the product $\mathbf{A}_i \bullet \exp(\mathbf{\Phi})$, where $\mathbf{\Phi}$ is some PSD matrix. The cost of our algorithm therefore depends on how the input is specified. When the input is given prefactored—that is, the m-by-m matrices $\mathbf{A}_i$'s are given as $\mathbf{A}_i = \mathbf{Q}_i \mathbf{Q}_i^\top$ and the matrix $\mathbf{C}^{-1/2}$ is given, then Theorem 4.1 can be used to compute matrix exponential in $O(\frac{1}{\varepsilon^3}(m+q) \log n \log q \log(1/\varepsilon))$ work and $O(\frac{1}{\varepsilon} \log n \log q \log(1/\varepsilon))$ depth, where q is the number of nonzero entries across $\mathbf{Q}_i$'s and $\mathbf{C}^{-1/2}$. This is because the matrix $\mathbf{\Phi}$ that we exponentiate has $\|\mathbf{\Phi}\|_2 \leq O(\frac{1}{\varepsilon} \log n)$, as shown in Lemma 3.8. Therefore, as a corollary to the main theorem, we have the following cost bounds:

Primal				*Dual*			
Minimize	$\mathsf{Tr}[\mathbf{Y}]$			Maximize	$\mathbf{1}^\top \boldsymbol{x}$		
Subject to:	$\mathbf{A}'_i \bullet \mathbf{Y}$	≥ 1	for $i = 1, \ldots, n$	Subject to:	$\sum_{i=1}^n x_i \mathbf{A}'_i$	$\preccurlyeq \mathbf{I}$	
	$\mathbf{Y}$	$\succcurlyeq 0$			$\boldsymbol{x}$	$\geq \mathbf{0}$.	

(1.2)

Figure 1. Normalized primal/dual positive SDPs. The symbol $\mathbf{I}$ represents the identity matrix.

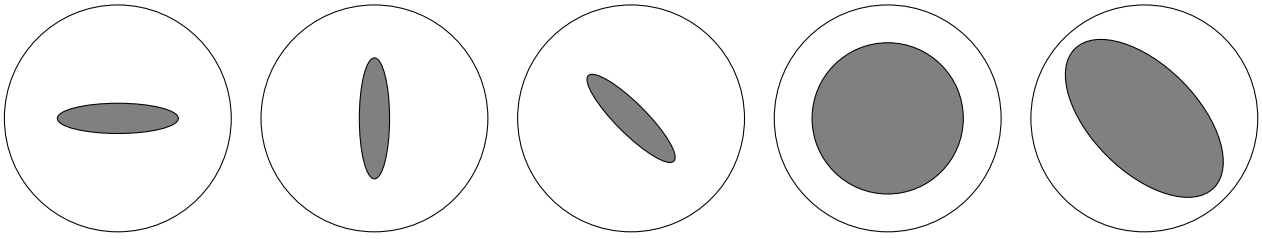

Figure 2. An instance of a packing SDP in 2 dimensions.

Corollary 1.2 *The algorithm* `approxPSDP` *has in* $\widetilde{O}(n+m+q)$ *work and* $O(\log^{O(1)}(n+m+q))$ *depth.*

If, however, the input program is not given in this form, we can add a preprocessing step that factors each $\mathbf{A}_i$ into $\mathbf{Q}_i\mathbf{Q}_i^\top$ since $\mathbf{A}_i$ is positive semidefinite. In general, this preprocessing requires at most $O(m^4)$ work and $O(\log^3 m)$ depth using standard parallel QR factorization [JáJ92]. Furthermore, these matrices often have certain structure that makes them easier to factor. Similarly, we can factor and invert $\mathbf{C}$ with the same cost bound, and can do better if it also has specialized structure.

2. BACKGROUND AND NOTATION

We review notation and facts that will prove useful later in the paper. Throughout this paper, we use the notation $\widetilde{O}(f(n))$ to mean $O(f(n)\,\mathrm{polylog}(f(n)))$.

Matrices and Positive Semidefiniteness. Unless otherwise stated, we will deal with real symmetric matrices in $\mathbb{R}^{m\times m}$. A symmetric matrix $\mathbf{A}$ is positive semidefinite, denoted by $\mathbf{A} \succcurlyeq \mathbf{0}$ or $\mathbf{0} \preccurlyeq \mathbf{A}$, if for all $\mathbf{z} \in \mathbb{R}^m$, $\mathbf{z}^\top\mathbf{A}\mathbf{z} \geq 0$. Equivalently, this means that all eigenvalues of $\mathbf{A}$ are nonnegative and the matrix $\mathbf{A}$ can be written as

$$\mathbf{A} = \sum_i \lambda_i \mathbf{v}_i \mathbf{v}_i^\top,$$

where $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ are the eigenvectors of $\mathbf{A}$ with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_m$ respectively. We will use $\lambda_1(\mathbf{A})$, $\lambda_2(\mathbf{A}), \ldots, \lambda_m(\mathbf{A})$ to represent the eigenvalues of $\mathbf{A}$ in decreasing order and also use $\lambda_{\max}(\mathbf{A})$ to denote $\lambda_1(\mathbf{A})$. Notice that positive semidefiniteness induces a partial ordering on matrices. We write $\mathbf{A} \preccurlyeq \mathbf{B}$ if $\mathbf{B} - \mathbf{A} \succcurlyeq \mathbf{0}$.

The trace of a matrix $\mathbf{A}$, denoted $\mathsf{Tr}[\mathbf{A}]$, is the sum of the matrix's diagonal entries: $\mathsf{Tr}[\mathbf{A}] = \sum_i A_{i,i}$. Alternatively, the trace of a matrix can be expressed as the sum of its eigenvalues, so $\mathsf{Tr}[\mathbf{A}] = \sum_i \lambda_i(\mathbf{A})$. Furthermore, we define

$$\mathbf{A} \bullet \mathbf{B} = \sum_{i,j} A_{i,j} B_{i,j} = \mathsf{Tr}[\mathbf{A}\mathbf{B}].$$

It follows that $\mathbf{A}$ is positive semidefinite if and only if $\mathbf{A}\bullet\mathbf{B} \geq 0$ for all PSD $\mathbf{B}$.

Matrix Exponential. Given an $m \times m$ symmetric positive semidefinite matrix $\mathbf{A}$ and a function $f : \mathbb{R} \to \mathbb{R}$, we define

$$f(\mathbf{A}) = \sum_{i=1}^m f(\lambda_i)\mathbf{v}_i\mathbf{v}_i^\top,$$

where, again, $\mathbf{v}_i$ is the eigenvector corresponding to the eigenvalue λ_i. It is not difficult to check that for $\exp(\mathbf{A})$, this definition coincides with $\exp(\mathbf{A}) = \sum_{i\geq 0} \frac{1}{i!}\mathbf{A}^i$.

Our algorithm relies on a matrix multiplicative weights (MMW) algorithm, which can be summarized as follows. For a fixed $\varepsilon_0 \leq \frac{1}{2}$ and $\mathbf{W}^{(1)} = \mathbf{I}$, we play a "game" a number of times, where in iteration $t = 1, 2, \ldots$, the following steps are performed:

1. Produce a probability matrix $\mathbf{P}^{(t)} = \mathbf{W}^{(t)} / \mathsf{Tr}\left[\mathbf{W}^{(t)}\right]$;
2. Incur a gain matrix $\mathbf{M}^{(t)}$; and
3. Update the weight matrix as

$$\mathbf{W}^{(t+1)} = \exp(\varepsilon_0 \sum_{t'\leq t} \mathbf{M}^{(t')}).$$

Like in the standard setting of multiplicative weight algorithms, the gain matrix is chosen by an external party, possibly adversarially. In our algorithm, the gain matrix is chosen to reflect the step we make in the iteration. Arora and Kale [AK07] shows that the MMW algorithm has the following guarantees (restated for our setting):

Theorem 2.1 ([AK07]) *For* $\varepsilon_0 \leq \frac{1}{2}$, *if* $\mathbf{M}^{(t)}$*'s are all PSD and* $\mathbf{M}^{(t)} \preccurlyeq \mathbf{I}$, *then after* T *iterations,*

$$(1+\varepsilon_0)\sum_{t=1}^T \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \geq \lambda_{\max}\left(\sum_{t=1}^T \mathbf{M}^{(t)}\right) - \frac{\ln n}{\varepsilon_0}. \quad (2.1)$$

3. SOLVING POSITIVE SDPS

In this section, we describe a parallel algorithm for solving positive packing SDPs, inspired by Young's algorithm for

positive LPs. As described earlier, by using binary search and appropriately scaling the input program, a positive packing SDP can be solved assuming an algorithm for the following decision problem:

> **Decision Problem:** Find either an $\boldsymbol{x} \in \mathbb{R}^+_n$ (a dual solution) such that
>
> $$\|\boldsymbol{x}\|_1 \geq 1 - \varepsilon \text{ and } \sum_{i=1}^{n} x_i \mathbf{A}_i \preccurlyeq \mathbf{I}$$
>
> or a PSD matrix $\mathbf{Y}$ (a primal solution) such that
>
> $$\mathsf{Tr}\left[\mathbf{Y}\right] \leq 1 \text{ and } \forall i, \mathbf{A}_i \bullet \mathbf{Y} \geq 1.$$

The following theorem provides a solution to this problem:

Theorem 3.1 *Let $0 < \varepsilon \leq 1$. There is an algorithm* `decisionPSDP` *that given a positive SDP, in $O\left(\frac{\log^3 n}{\varepsilon^4}\right)$ iterations solves the Decision Problem.*

Presented in Algorithm 3.1 is an algorithm that satisfies the theorem. But before we go about proving it, let us take a closer look at the algorithm. Fix an accuracy parameter $\varepsilon > 0$. We set K to $\frac{1}{\varepsilon}(1 + \ln n)$. The reason for choosing this value is technical, but the motivation was so that we can absorb the $\ln n$ term in Theorem 2.1 and account for the contribution of the starting solution $\boldsymbol{x}^{(0)}$.

The algorithm is a multiplicative weight updates algorithm, which proceeds in rounds. Initially, we work with the starting solution $x_i^{(0)} = \frac{1}{n\,\mathsf{Tr}[\mathbf{A}_i]}$. This solution is chosen to be small so that $\sum_i x_i^{(0)} \mathbf{A}_i \preccurlyeq \mathbf{I}$, hence respecting the dual constraint, and it contains enough mass that subsequent updates (each update is a multiple of the current solution) are guaranteed to make rapid progress. In each iteration, we compute $\mathbf{W}^{(t)} = \exp(\mathbf{\Psi}^{(t-1)})$, where $\mathbf{\Psi}^{(t-1)} = \sum_i x_i^{(t-1)} \mathbf{A}_i$. (For some intuitions for the $\mathbf{W}^{(t)}$ matrix, we refer the reader to [AK07, Kal07].)

The next two steps (Steps 3–4) of the algorithm are responsible for identifying which coordinates of $\boldsymbol{x}$ to update. For starters, it may help to think of them as follows: let $\mathbf{P}^{(t)} = \mathbf{W}^{(t)} / \mathsf{Tr}\left[\mathbf{W}^{(t)}\right]$—and $B^{(t)}$ be $\{i \in [n] : \mathbf{P}^{(t)} \bullet \mathbf{A}_i \leq 1 + \varepsilon\}$. The actual algorithm discretizes $\mathsf{Tr}\left[\mathbf{W}^{(t)}\right]$ to ensure certain monotonicity properties on $B^{(t)}$. As we show later on in Lemma 3.2, the set $B^{(t)}$ cannot be empty unless the system is infeasible. The final steps of the algorithm increment each coordinate $i \in B^{(t)}$ of the solution by the amount $\alpha \cdot x_i^{(t)}$.

The choice of α may seem mysterious at this point; it is chosen to ensure that (1) $\sum_i \delta_i^{(t)} \mathbf{A}_i \preccurlyeq \varepsilon \mathbf{I}$ and (2) $\mathbf{1}^\top \delta^{(t)} \leq \varepsilon$. Intuitively, these bounds prevent us from taking too big a step from the current solution. At a more technical level, the first requirement is needed to satisfy the MMW algorithm's condition, and the second requirement makes sure when we cannot overshoot by much when exiting from the while loop.

3.1 Analysis

We will bound the approximation guarantees and analyze the cost of the algorithm. Before we start, we will need some notation and definitions. An easy induction gives that the quantities that we track across the iterations of Algorithm 3.1 satisfy the following relationships:

Algorithm 3.1 Parallel Packing SDP algorithm

Let $K = \frac{1}{\varepsilon}(1 + \ln n)$.
Let $x_i^{(0)} = \frac{1}{n \cdot \mathsf{Tr}[\mathbf{A}_i]}$.
Initialize $\mathbf{\Psi}^{(0)} = \sum_{i=1}^n x_i^{(0)} \mathbf{A}_i$, $t = 0$.
While $\|\boldsymbol{x}^{(t)}\|_1 \leq K$

1. $t = t + 1$.
2. Let $\mathbf{W}^{(t)} = \exp(\mathbf{\Psi}^{(t-1)})$.
3. Let p be such that $(1+\varepsilon)^{p-1} < \mathsf{Tr}\left[\mathbf{W}^{(t)}\right] \leq (1+\varepsilon)^p$.
4. Let $B^{(t)} = \{i \in [n] : \mathbf{W}^{(t)} \bullet \mathbf{A}_i \leq (1+\varepsilon)^{p+1}\}$.
5. If $B^{(t)}$ is empty, return $\mathbf{Y}^* = \mathbf{W}^{(t)} / \mathsf{Tr}\left[\mathbf{W}^{(t)}\right]$ as a primal solution.
6. Let $\delta^{(t)} = \alpha \cdot \boldsymbol{x}_B^{(t-1)}$, where
$$\alpha = \min\{\varepsilon / \|\boldsymbol{x}_B^{(t-1)}\|_1, \varepsilon/(1+10\varepsilon)K\}.$$
7. Update $\boldsymbol{x}^{(t)} = \boldsymbol{x}^{(t-1)} + \delta^{(t)}$ and $\mathbf{\Psi}^{(t)} = \mathbf{\Psi}^{(t-1)} + \sum_{i=1}^n \delta^{(t)} \mathbf{A}_i$

Return $\boldsymbol{x}^* = \frac{1}{K(1+10\varepsilon)} \boldsymbol{x}^{(t)}$ as a dual solution.

$$\boldsymbol{x}^{(t)} = \boldsymbol{x}^{(0)} + \sum_{\tau=1}^{t} \delta^{(\tau)} \tag{3.1}$$

$$\mathbf{W}^{(t)} = \exp(\mathbf{\Psi}^{(t-1)}) \tag{3.2}$$

$$\mathbf{P}^{(t)} \stackrel{\text{def}}{=} \mathbf{W}^{(t)} / \mathsf{Tr}\left[\mathbf{W}^{(t)}\right] \tag{3.3}$$

$$\begin{aligned} \mathsf{Tr}\left[\mathbf{P}^{(t)}\right] &= \mathsf{Tr}\left[\mathbf{W}^{(t)} / \mathsf{Tr}\left[\mathbf{W}^{(t)}\right]\right] \\ &= \mathsf{Tr}\left[\mathbf{W}^{(t)}\right] / \mathsf{Tr}\left[\mathbf{W}^{(t)}\right] = 1 \end{aligned} \tag{3.4}$$

$$\mathbf{M}^{(t)} \stackrel{\text{def}}{=} \frac{1}{\varepsilon} \sum_{i=1}^{n} \delta_i^{(t)} \mathbf{A}_i \qquad \text{when } t \geq 1 \tag{3.5}$$

$$\mathbf{\Psi}^{(t)} = \sum_{i=1}^{n} x_i^{(t)} \mathbf{A}_i = \varepsilon \sum_{\tau=0}^{t} \mathbf{M}^{(\tau)} \tag{3.6}$$

To bound the approximation guarantees and the cost of this algorithm, we reason about the spectrum of $\mathbf{\Psi}^{(t)}$ and the ℓ_1 norm of the vector $\boldsymbol{x}^{(t)}$ as the algorithm executes. Since the coordinates of our vector $\boldsymbol{x}^{(t)}$ are always non-negative, we note that $\|\boldsymbol{x}^{(t)}\|_1 = \mathbf{1}^\top \boldsymbol{x}^{(t)}$ and we use either notation as convenient. We begin the analysis by showing that $B^{(t)}$ can never be empty unless the system is infeasible:

Lemma 3.2 (Feasibility) *If there is an iteration t such that $B^{(t)}$ is empty, then $\mathbf{P}^{(t)} = \mathbf{W}^{(t)} / \mathsf{Tr}\left[\mathbf{W}^{(t)}\right]$ is a valid primal solution with objective value 1. Furthermore, by duality theory, there exists no dual solution $\boldsymbol{x} \in \mathbb{R}^n_+$ with objective value at least 1.*

Proof. The fact that $B^{(t)}$ is empty means that for all $i = 1, \ldots, n$, $\mathbf{W}^{(t)} \bullet \mathbf{A}_i \geq (1+\varepsilon)^{p+1}$. But we know that $\mathsf{Tr}\left[\mathbf{W}^{(t)}\right] > (1+\varepsilon)^{p-1}$, so

$$\mathbf{P}^{(t)} \bullet \mathbf{A}_i = \frac{\mathbf{W}^{(t)}}{\mathsf{Tr}\left[\mathbf{W}^{(t)}\right]} \bullet \mathbf{A}_i \geq (1+\varepsilon)^2 \geq 1,$$

As noted above, $\mathsf{Tr}\left[\mathbf{P}^{(t)}\right] = 1$, so $\mathbf{P}^{(t)}$ is a valid primal solution with objective at most 1, so no dual solution with objective more than 1 exists. □

We proceed to analyze the vector $\boldsymbol{x}^{(t)}$ in the case that $B^{(t)}$ never becomes empty. The main loop in Algorithm 3.1 terminates only if $\|\boldsymbol{x}^{(t)}\|_1 > K$, so the solution we produce satisfies

$$\|\boldsymbol{x}^*\|_1 = \frac{1}{(1+10\varepsilon)K}\|\boldsymbol{x}^{(t)}\|_1 \geq \frac{K}{(1+10\varepsilon)K} \geq 1-10\varepsilon \quad (3.7)$$

In order for this to be a dual solution, we still need to show that it satisfies $\sum_i x_i^* \mathbf{A}_i \preccurlyeq \mathbf{I}$. In particular, it suffices to show that $\frac{1}{(1+10\varepsilon)K}\mathbf{\Psi}^{(T)} \preccurlyeq \mathbf{I}$, where T is the final iteration.

Spectrum Bounds. The rest of the analysis hinges on bounding the spectrum of $\mathbf{\Psi}^{(t)}$. More specifically, we prove the following lemma, which shows that the spectrum of all intermediate $\mathbf{\Psi}^{(t)}$'s is bounded by $(1+O(\varepsilon))K$:

Lemma 3.3 (Spectrum Bound) *For $t = 0, \ldots, T$, where T is the final iteration,*

$$\mathbf{\Psi}^{(t)} = \sum_{i=1}^{n} x_i^{(t)} \mathbf{A}_i \preccurlyeq (1+10\varepsilon)K\mathbf{I}. \quad (3.8)$$

We prove this lemma by resorting to properties of the MMW algorithm (Theorem 2.1), which relates the final spectral values to the "gain" derived at each intermediate step. For this, we will show a claim (Claim 3.5) that quantifies the gain we get in each step as a function of the ℓ_1-norm of the change we make in that step. But first, we analyze the initial matrix:

Claim 3.4

$$\lambda_{\max}\left(\mathbf{\Psi}^{(0)}\right) = \lambda_{\max}\left(\sum_{i=1}^{n} x_i^{(0)}\mathbf{A}_i\right) \leq 1$$

Proof. Our choice of $\boldsymbol{x}^{(0)}$ guarantees that for all $i = 1, \ldots, n$,

$$x_i^{(0)}\mathbf{A}_i = \frac{1}{n\,\mathsf{Tr}\left[\mathbf{A}_i\right]}\mathbf{A}_i \preccurlyeq \frac{1}{n}\mathbf{I}.$$

Summing across $i = 1, \ldots, n$ gives the desired bound. □

The following claim bounds the value of $\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)}$ in terms of the ℓ_1 norm of the change to the dual solution vector. This is precisely the quantity we track in Theorem 2.1:

Claim 3.5 *For all $t = 1, \ldots, T$,*

$$\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \leq \frac{(1+\varepsilon)^2}{\varepsilon} \cdot \|\delta^{(t)}\|_1. \quad (3.9)$$

Proof. Consider that

$$\begin{aligned}\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} &= \frac{1}{\varepsilon}\left(\sum_{i=1}^{n} \delta_i^{(t)}\mathbf{A}_i\right) \bullet \mathbf{P}^{(t)} \\ &= \frac{1}{\varepsilon}\left(\sum_{i\in B} \delta_i^{(t)}\mathbf{A}_i \bullet \mathbf{P}^{(t)}\right)\end{aligned}$$

Every $i \in B^{(t)}$ has the property that

$$\mathbf{A}_i \bullet \mathbf{W}^{(t)} \leq (1+\varepsilon)^{p+1}$$

So then, since

$$\mathbf{P}^{(t)} = \frac{\mathbf{W}^{(t)}}{\mathsf{Tr}\left[\mathbf{W}^{(t)}\right]} \preccurlyeq \frac{\mathbf{W}^{(t)}}{(1+\varepsilon)^{p-1}},$$

we have

$$\mathbf{A}_i \bullet \mathbf{P}^{(t)} \leq (1+\varepsilon)^2$$

and thus

$$\begin{aligned}\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} &\leq \frac{1}{\varepsilon}\left(\sum_{i\in B} \delta_i^{(t)}(1+\varepsilon)^2\right) \\ &\leq \frac{(1+\varepsilon)^2}{\varepsilon}\|\delta^{(t)}\|_1,\end{aligned}$$

which proves the claim. □

We can then bound the total ℓ_1 norm of the change to the dual solution by bounding the ℓ_1 norm of $\boldsymbol{x}^{(t)}$. Specifically we show that unless the algorithm terminates at the first iteration, $\boldsymbol{x}^{(T)}$ does not exceed $K+\varepsilon$ in ℓ_1 norm.

Claim 3.6 *For $t = 1, \ldots, T$,*

$$\|\boldsymbol{x}^{(t)}\|_1 \leq (1+\varepsilon)K$$

Proof. Since T is the iteration when the algorithm terminate and for all $t \geq 0$, $\delta^{(t)} \in \mathbb{R}^n_+$, we have

$$\begin{aligned}\|\boldsymbol{x}^{(t)}\|_1 &\leq \|\boldsymbol{x}^{(t-1)}\|_1 + \|\delta^{(t)}\|_1 \\ &\leq K + \|\delta^{(t)}\|_1\end{aligned}$$

By our choice of α, we know that $\alpha \leq \varepsilon/\|\boldsymbol{x}_B^{(t-1)}\|_1$ and therefore $\|\delta^{(T)}\|_1 = \alpha\|\boldsymbol{x}_B^{(t-1)}\|_1 \leq \varepsilon$. Substituting this into the equation above gives $\|\boldsymbol{x}^{(t)}\|_1 \leq K+\varepsilon$ and the claim follows from $K \geq 1$. □

We are now ready to complete the proof of the spectrum bound lemma (Lemma 3.3):

Proof of Lemma 3.3. We can rewrite $\mathbf{\Psi}^{(t)}$ as

$$\mathbf{\Psi}^{(t)} = \sum_{i=1}^{n} x_i^{(0)}\mathbf{A}_i + \sum_{\tau=1}^{t}\sum_{i=1}^{n} \delta_i^{(\tau)}\mathbf{A}_i = \sum_{i=1}^{n} x_i^{(0)}\mathbf{A}_i + \varepsilon\sum_{\tau=1}^{t}\mathbf{M}^{(\tau)},$$

so

$$\lambda_{\max}(\mathbf{\Psi}^{(t)}) \leq \lambda_{\max}\left(\sum_{i=1}^{n} x_i^{(0)}\mathbf{A}_i\right) + \varepsilon \cdot \lambda_{\max}\left(\sum_{\tau=1}^{t}\mathbf{M}^{(\tau)}\right)$$

since both sums yield positive semidefinite matrices.

By Claim 3.4, we know that the first term is at most 1. To bound the second term, we again apply Theorem 2.1, which we restate below:

$$(1+\varepsilon)\sum_{\tau=1}^{t}\mathbf{M}^{(\tau)} \bullet \mathbf{P}^{(\tau)} \geq \lambda_{\max}\left(\sum_{\tau=1}^{t}\mathbf{M}^{(t)}\right) - \frac{\ln n}{\varepsilon}$$

Rearranging terms, we have

$$\lambda_{\max}\left(\sum_{\tau=1}^{t}\mathbf{M}^{(\tau)}\right) \leq (1+\varepsilon)\sum_{\tau=1}^{t}\mathbf{M}^{(\tau)} \bullet \mathbf{P}^{(\tau)} + \frac{\ln n}{\varepsilon}.$$

We also want to make sure that each $\mathbf{M}^{(\tau)}$ satisfies $\mathbf{M}^{(\tau)} \preccurlyeq \mathbf{I}$. With an easy induction on t, we can show that (3.8) holds

for all $\tau \le t-1$. This means that for $\tau = 1, \dots, t$, each $\mathbf{M}^{(\tau)}$ satisfies

$$\begin{aligned}\mathbf{M}^{(\tau)} &= \frac{1}{\varepsilon}\sum_{i=1}^{n}\delta_i^{(\tau)}\mathbf{A}_i \\ &\preccurlyeq \frac{\alpha}{\varepsilon}\sum_{i=1}^{n}x_i^{(\tau)}\mathbf{A}_i \\ &\preccurlyeq \frac{\varepsilon/(1+10\varepsilon)K}{\varepsilon}\sum_{i=1}^{n}x_i^{(\tau)}\mathbf{A}_i \\ &\preccurlyeq \frac{\varepsilon/(1+10\varepsilon)K}{\varepsilon}(1+10\varepsilon)K\mathbf{I} \preccurlyeq \mathbf{I},\end{aligned}$$

since $\alpha \le \varepsilon/(1+10\varepsilon)K$. It then follow from Claim 3.5 that

$$\begin{aligned}\varepsilon\cdot\lambda_{\max}\left(\sum_{\tau=1}^{t}\mathbf{M}^{(\tau)}\right) &\le (1+\varepsilon)\sum_{\tau=1}^{t}(1+\varepsilon)^2\cdot\|\delta^{(\tau)}\|_1 + \ln n \\ &= \ln n + (1+\varepsilon)^3\|\boldsymbol{x}^{(t)}\|_1\end{aligned}$$

Where the last step follows from $\boldsymbol{x}^{(t)} = \boldsymbol{x}^{(0)} + \sum_{\tau=1}^{t}\delta^{(\tau)}$ and each entry of $\boldsymbol{x}^{(0)}$ and $\delta^{(\tau)}$ being non-negative. Applying the bound on $\|\boldsymbol{x}^{(t)}\|_1$ from Claim 3.6 then gives:

$$\varepsilon\cdot\lambda_{\max}\left(\sum_{\tau=1}^{t}\mathbf{M}^{(\tau)}\right) \le \ln n + (1+\varepsilon)^4 K$$

Putting these together, we get:

$$\lambda_{\max}(\mathbf{\Psi}^{(t)}) \le 1 + \ln n + (1+\varepsilon)^4 K \le \varepsilon K + (1+\varepsilon)^4 K,$$

which allows us to conclude that $\mathbf{\Psi}^{(t)} \preccurlyeq (1+10\varepsilon)K\mathbf{I}$. □

The spectrum bound lemma says that at any point in the algorithm, the solution vector $\boldsymbol{x}^{(t)}$ satisfies $\sum_i x_i^{(t)}\mathbf{A}_i \preccurlyeq (1+10\varepsilon)K\mathbf{I}$. Together with Equation (3.7), we know that if the algorithm completes the **while** loop, the solution $\boldsymbol{x}^*$ that we return satisfies $\|\boldsymbol{x}^*\|_1 \ge 1-10\varepsilon$ and

$$\sum_i x_i^*\mathbf{A}_i = \frac{1}{(1+10\varepsilon)K}\sum_i x_i^{(t)}\mathbf{A}_i \preccurlyeq \mathbf{I}.$$

Thus, $\boldsymbol{x}^*$ is indeed a dual solution with value at least $1-10\varepsilon$.

To piece everything together, we set ε to $\varepsilon/10$, so if there is an iteration in which $B^{(t)}$ is empty, we produce a primal solution with value at most 1; otherwise, we return a dual solution with value at least $1-\varepsilon$. Hence, the algorithm has the promised approximation bounds. Next we will analyze its cost.

Cost Analysis. Similar to Young's analysis, our analysis relies on the notion of phases, grouping together iterations with similar $\mathbf{W}^{(t)}$ matrices into a phase in a way that ensures the existence of a coordinate i with the property that this coordinate is incremented (i.e., $\delta_i^{(t)} > 0$) by a significant amount in every iteration of this phase. To this end, we say that an iteration t belongs to *phase* p if and only if $(1+\varepsilon)^{p-1} < \mathsf{Tr}\left[\mathbf{W}^{(t)}\right] \le (1+\varepsilon)^p$. A phase ends when the algorithm terminates or the next iteration belongs to a different phase.

Almost immediate from this definition is a bound on the number of phases:

Lemma 3.7 *The total number of phases is at most $O(K/\varepsilon)$.*

Proof. On the one hand, we have $\mathbf{0} \preccurlyeq \mathbf{\Psi}^{(0)}$, so

$$\mathsf{Tr}\left[\mathbf{W}^{(0)}\right] \ge n\cdot e^0 = n.$$

On the other hand, we know that $\mathbf{\Psi}^{(T)} \preccurlyeq (1+O(\varepsilon))K$, so $\mathsf{Tr}\left[\mathbf{W}^{(t)}\right] \le n\exp\left((1+O(\varepsilon))K\right)$. This means that the total of number is phases is at most

$$\begin{aligned}\log_{1+\varepsilon}\frac{\mathsf{Tr}\left[\mathbf{W}^{(T)}\right]}{\mathsf{Tr}\left[\mathbf{W}^{(0)}\right]} &\le \log_{1+\varepsilon}\exp\left((1+O(\varepsilon))K\right) &(3.10)\\ &\le \frac{1}{\varepsilon}(1+O(\varepsilon))K = O\left(\frac{K}{\varepsilon}\right) &(3.11)\end{aligned}$$

□

To bound the total number of steps, we'll analyze the number of iterations within a phase. For this, we'll need a couple of claims. The first claim shows that if a coordinate is incremented at the end of a phase, it must have been incremented at every iteration of this phase.

Claim 3.8 *If $i \in B^{(t)}$, then for all $t' < t$ belonging to the same phase as t, $i \in B^{(t')}$.*

Proof. Suppose t belongs to phase p. As $i \in B^{(t)}$, we know that $\mathbf{W}^{(t)} \bullet \mathbf{A}_i \le (1+\varepsilon)^{p+1}$. Since $t' < t$ we have

$$\begin{aligned}\mathbf{\Psi}^{(t)} - \mathbf{\Psi}^{(t')} &= \sum_{t'\le\tau<t}\mathbf{M}^{(\tau)} &(3.12)\\ &= \sum_{t'\le\tau<t}\sum_i \delta_i^{(\tau)}\mathbf{A}_i \succeq 0 &(3.13)\end{aligned}$$

Therefore $\mathbf{W}^{(t')} \preccurlyeq \mathbf{W}^{(t)}$ and that

$$\mathbf{W}^{(t')} \bullet \mathbf{A}_i \le \mathbf{W}^{(t)} \bullet \mathbf{A}_i \le (1+\varepsilon)^{p+1}, \tag{3.14}$$

which means that $i \in B^{(t')}$, as desired. □

In the second claim, we'll place a bounding box around each coordinate $x_i^{(t)}$ of our solution vectors. This turns out to be an important machinery in bounding the number of iterations required by the algorithm.

Claim 3.9 (Bounding Box) *For all index i, at any iteration t,*

$$x_i^{(t)} \le (1+O(\varepsilon))n^2K/\,\mathsf{Tr}\left[\mathbf{A}_i\right]x_i^{(0)}$$

Proof. Recall that $x_i^{(0)} = 1/(n\,\mathsf{Tr}\left[\mathbf{A}_i\right])$. To argue an upper bound on $x_i^{(t)}$, note that Lemma 3.3 gives

$$\mathbf{\Psi}^{(t)} \preccurlyeq (1+O(\varepsilon))\mathbf{I}K \tag{3.15}$$

Since $\sum_{j=1}^{n}x_j^{(t)}\mathbf{A}_i = \mathbf{\Psi}^{(t)}$ and each $\mathbf{A}_i$ is positive semidefinite, we have

$$\mathsf{Tr}\left[x_i^{(t)}\mathbf{A}_i\right] \le \mathsf{Tr}\left[\mathbf{\Psi}^{(t)}\right] \le (1+O(\varepsilon))nK \tag{3.16}$$

We conclude that $x_i^{(t)} \le (1+O(\varepsilon))n^2K/\,\mathsf{Tr}\left[\mathbf{A}_i\right]x_i^{(0)}$, as claimed. □

The final claim shows that each iteration makes significant progress in incrementing the solution.

Claim 3.10 *In each iteration, either $\|\delta^{(t)}\|_1 = \varepsilon$ or $\alpha \ge \Omega(\varepsilon/K)$.*

PROOF. We chose α to be $\min\{\varepsilon/\|\boldsymbol{x}_B^{(t-1)}\|_1, \varepsilon/(1+10\varepsilon)K\}$. If $\alpha = \varepsilon/\|\boldsymbol{x}_B^{(t-1)}\|_1$, then $\|\delta^{(t)}\|_1 = \varepsilon$ and we are done. Otherwise, we have $\alpha = \varepsilon/(1+10\varepsilon)K$, which is $\Omega(\varepsilon/K)$. □

Combining these claims, we have the following bounds on the number of iterations:

Corollary 3.11 *The number of iterations per phase is at most*

$$O\left(\frac{K}{\varepsilon}\ln(nK)\right).$$

and the total number of iterations is at most:

$$O\left(\frac{\log^3 n}{\varepsilon^4}\right)$$

PROOF. Consider a phase of the algorithm, and let f be the final iteration of this phase. By Claim 3.10, each iteration t satisfies $\|\delta^{(t)}\|_1 = \varepsilon$ or $\alpha \geq \Omega(\varepsilon/K)$. Since $\|\boldsymbol{x}^{(t)}\|_1 \leq K + \varepsilon$ for all $t \leq T$, the number of iterations in which $\|\delta^{(t)}\|_1 = \varepsilon$ can be at most $O(K/\varepsilon)$. Now, let $i \in B^{(f)}$ be a coordinate that got incremented in the final iteration of this phase. By Claim 3.8, this coordinate got incremented in every iteration of this phase. Therefore, the number of iterations within this phase where $\alpha \geq \Omega(\varepsilon/K)$ is at most

$$\begin{aligned}\log_{1+\Omega(\varepsilon/K)}(x_i^{(f)}/x^{(0)}) &\leq \log_{1+\Omega(\varepsilon/K)}\left((1+O(\varepsilon)n^2K\right)\\ &= O\left(\frac{K}{\varepsilon}\ln\left((1+O(\varepsilon))K\right)\right).\end{aligned}$$

Combining with Lemma 3.7 and the setting of $K = O\left(\frac{\log n}{\varepsilon}\right)$ gives the overall bound. □

4. MATRIX EXPONENTIAL EVALUATION

We describe a fast algorithm for computing the matrix dot product of a positive semidefinite matrix and the matrix exponential of another positive semidefinite matrix.

Theorem 4.1 *There is an algorithm* `bigDotExp` *that when given a m-by-m matrix $\boldsymbol{\Phi}$ with p non-zero entries, $\kappa \geq \max\{1, \|\boldsymbol{\Phi}\|_2\}$, and m-by-m matrices $\mathbf{A}_i$ in factorized form $\mathbf{A}_i = \mathbf{Q}_i\mathbf{Q}_i^\top$ where the total number of nonzeros across all $\mathbf{Q}_i$ is q;* `bigDotExp`$(\boldsymbol{\Phi}, \{\mathbf{A}_i = \mathbf{Q}_i\mathbf{Q}_i^\top\}_{i=1}^n)$ *computes $(1\pm\varepsilon)$ approximations to all $\exp(\boldsymbol{\Phi})\bullet\mathbf{A}_i$ in $O(\kappa\log m\log(1/\varepsilon))$ depth and $O(\frac{1}{\varepsilon^2}(\kappa\log(1/\epsilon)p+q)\log m)$ work.*

The idea behind Theorem 4.1 is to approximate the matrix exponential using a low-degree polynomial because evaluating matrix exponentials exactly is costly. For this, we will apply the following lemma, reproduced from Lemma 6 in [AK07]:

Lemma 4.2 ([AK07]) *If $\mathbf{B}$ is a PSD matrix such that $\|\mathbf{B}\|_2 \leq \kappa$, then the operator*

$$\widehat{\mathbf{B}} = \sum_{0\leq i<k}\frac{1}{i!}\mathbf{B}^i \qquad \text{where } k = \max\{e^2\kappa, \ln(2\varepsilon^{-1})\}$$

satisfies

$$(1-\varepsilon)\exp(\mathbf{B}) \preceq \widehat{\mathbf{B}} \preceq \exp(\mathbf{B}).$$

PROOF OF THEOREM 4.1. The given factorization of each $\mathbf{A}_i$ allows us to write $\exp(\boldsymbol{\Phi})\bullet\mathbf{A}_i$ as the 2-norm of a vector:

$$\begin{aligned}\exp(\boldsymbol{\Phi})\bullet\mathbf{A}_i &= \mathsf{Tr}\left[\exp(\boldsymbol{\Phi})\mathbf{Q}_i\mathbf{Q}_i^\top\right]\\ &= \mathsf{Tr}\left[\mathbf{Q}_i^\top\exp(\tfrac{1}{2}\boldsymbol{\Phi})\exp(\tfrac{1}{2}\boldsymbol{\Phi})\mathbf{Q}_i\right]\\ &= \|\exp(\tfrac{1}{2}\boldsymbol{\Phi})\mathbf{Q}_i\|_2\end{aligned}$$

By Lemma 4.2, it suffices to evaluate $\widehat{\mathbf{B}}\bullet\mathbf{A}_i$ where $\widehat{\mathbf{B}}$ is an approximation to $\mathbf{B} = \exp(\frac{1}{2}\boldsymbol{\Phi})$. To further reduce the work, we can apply the Johnson-Lindenstrauss transformation [DG03, IM98] to reduce the length of the vectors to $O(\log m)$; specifically, we find a $O(\frac{1}{\varepsilon^2}\log m)\times m$ Gaussian matrix $\boldsymbol{\Pi}$ and evaluate

$$\|\boldsymbol{\Pi}\widehat{\mathbf{B}}\mathbf{Q}_i\|_2$$

Since $\boldsymbol{\Pi}$ only has $O(\frac{1}{\varepsilon^2}\log m)$ rows, we can compute $\boldsymbol{\Pi}\widehat{\mathbf{B}}$ using $O(\log m)$ evaluations of $\widehat{\mathbf{B}}$. The work/depth bounds follow from doing each of the evaluations of $\widehat{\mathbf{B}}\Pi_i$, where Π_i denotes the i-th column of $\boldsymbol{\Pi}$, and matrix-vector multiplies involving $\boldsymbol{\Phi}$ in parallel. □

5. CONCLUSION

We presented a simple NC parallel algorithm for packing SDPs that requires $O(\frac{1}{\varepsilon^4}\log^4 n\log(\frac{1}{\varepsilon}))$ iterations, where each iteration involves only simple matrix operations and computing the trace of the product of a matrix exponential and a positive semidefinite matrix. When a positive SDP is given in a factorized form, we showed how the dot product with matrix exponential can be implemented in nearly-linear work, leading to an algorithm with $\widetilde{O}(m+n+q)$ work, where n is the number of constraint matrices, m is the dimension of these matrices, and q is the total number of nonzero entries in the factorization.

Compared to the situation with positive LPs, the classification of positive SDPs is much richer because packing/covering constraints can take many forms, either as matrices (e.g. $\sum_{i=1}^n x_i\mathbf{A}_i \preccurlyeq \mathbf{I}$ for packing, $\sum_{i=1}^n x_i\mathbf{A}_i \succcurlyeq \mathbf{I}$ for covering) or as dot products between matrices (e.g. $\mathbf{A}_i\bullet\mathbf{Y} \leq 1$ for packing, $\mathbf{A}_i\bullet\mathbf{Y} \geq 1$ for covering). The positive SDPs studied in [JY11] and our paper should be compared with the closely related notion of covering SDPs studied by Iyengar et al [IPS10]; however, among the applications they examine, only the beamforming SDP relaxation discussed in Section 2.2 of [IPS10] falls completely within the framework of packing SDPs as defined in 1.2. Problems such as MAXCUT and SPARSESTCUT require additional matrix-based packing constraints. We believe extending these algorithms to solve mixed packing/covering SDPs is an interesting direction for future work.

Acknowledgments

This work is partially supported by the National Science Foundation under grant numbers CCF-1018463, CCF-1018188, and CCF-1016799 and by generous gifts from IBM, Intel, and Microsoft. Richard Peng is supported by a Microsoft Fellowship. We thank the SPAA reviewers for suggestions that helped improve this paper.

References

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semide.nite programming using the multiplicative weights update method. In *FOCS*, pages 339–348, 2005.

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007.

[DG03] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003.

[GK98] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Symposium on the Foundations of Computer Science (FOCS)*, pages 300–309, 1998.

[GLS93] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, New York, 2nd edition, 1993.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC)*, pages 604–613, 1998.

[IPS10] Garud Iyengar, David J. Phillips, and Clifford Stein. Feasible and accurate algorithms for covering semidefinite programs. In *SWAT*, pages 150–162, 2010.

[IPS11] Garud Iyengar, David J. Phillips, and Clifford Stein. Approximating semidefinite packing programs. *SIAM Journal on Optimization*, 21(1):231–268, 2011.

[JáJ92] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[JY11] Rahul Jain and Penghui Yao. A parallel approximation algorithm for positive semidefinite programming. In *FOCS*, pages 463–471, 2011.

[JY12] Rahul Jain and Penghui Yao. A parallel approximation algorithm for mixed packing and covering semidefinite programs. *CoRR*, abs/1201.6090, 2012.

[Kal07] Satyen Kale. *Efficient Algorithms using the Multiplicative Weights Update Method*. PhD thesis, Princeton University, August 2007. Princeton Tech Report TR-804-07.

[KL96] Philip N. Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *STOC*, pages 338–347, 1996.

[KY07] Christos Koufogiannakis and Neal E. Young. Beating simplex for fractional packing and covering linear programs. In *FOCS*, pages 494–504, 2007.

[KY09] Christos Koufogiannakis and Neal E. Young. Distributed and parallel algorithms for weighted vertex cover and other covering problems. In *PODC*, pages 171–179, 2009.

[LN93] Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *STOC'93*, pages 448–457, New York, NY, USA, 1993.

[PST95] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.*, 20(2):257–301, 1995.

[You01] Neal E. Young. Sequential and parallel algorithms for mixed packing and covering. In *FOCS*, pages 538–546, 2001.

APPENDIX

A. NORMALIZED POSITIVE SDPS

This is the same transformation that Jain and Yao presented [JY11]; we only present it here for easy reference.

Consider the primal program in (1.1). It suffices to show that it can be transformed into the following program without changing the optimal value:

$$\begin{array}{lll} \text{Minimize} & \mathsf{Tr}\left[\mathbf{Z}\right] & \\ \text{Subject to:} & \mathbf{B}_i \bullet \mathbf{Z} \geq 1 & \text{for } i = 1, \ldots, m \\ & \mathbf{Z} \succcurlyeq \mathbf{0}, & \end{array} \tag{A.1}$$

We can make the following assumptions without loss of generality: First, $b_i > 0$ for all $i = 1, \ldots, m$ because if b_i were 0, we could have thrown it away. Second, all $\mathbf{A}_i$'s are the support of $\mathbf{C}$, or otherwise we know that the corresponding dual variable must be set to 0 and therefore can be removed right away. Therefore, we will treat C as having a full-rank, allowing us to define

$$\mathbf{B}_i \stackrel{\text{def}}{=} \frac{1}{b_i}\mathbf{C}^{-1/2}\mathbf{A}_i\mathbf{C}^{-1/2}$$

It is not hard to verify that the normalized program (A.1) has the same optimal value as the original SDP (1.1).

Note that if we're given factorization of $\mathbf{A}_i$ into $\mathbf{Q}_i\mathbf{Q}_i^\top$, then $\mathbf{B}_i$ can also be factorized as:

$$\mathbf{B}_i = \frac{1}{b_i}(\mathbf{C}^{-1/2}\mathbf{Q}_i)(\mathbf{C}^{-1/2}\mathbf{Q}_i)^\top$$

Furthermore, it can be checked that the dual of the normalized program is the same as the dual in Equation 1.2.