Online Semidefinite Programming*

Noa Elad[1], Satyen Kale[2], and Joseph (Seffi) Naor[3]

1 **Computer Science Dept., Technion, Haifa, Israel**
noako@cs.technion.ac.il
2 **Yahoo Research, New York, NY, USA**
satyen@yahoo-inc.com
3 **Computer Science Dept., Technion, Haifa, Israel**
naor@cs.technion.ac.il

Abstract

We consider semidefinite programming through the lens of online algorithms – what happens if not all input is given at once, but rather iteratively? In what way does it make sense for a semidefinite program to be revealed? We answer these questions by defining a model for online semidefinite programming. This model can be viewed as a generalization of online covering-packing linear programs, and it also captures interesting problems from quantum information theory. We design an online algorithm for semidefinite programming, utilizing the online primal-dual method, achieving a competitive ratio of $O(\log n)$, where n is the number of matrices in the primal semidefinite program. We also design an algorithm for semidefinite programming with box constraints, achieving a competitive ratio of $O(\log F^*)$, where F^* is a sparsity measure of the semidefinite program. We conclude with an online randomized rounding procedure.

1998 ACM Subject Classification G.1.6 Optimization

Keywords and phrases online algorithms, semidefinite programming, primal-dual

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.40

1 Introduction

The study of online algorithms is a major theme in computer science. In contrast to an algorithm that receives all its input at once, an online algorithm receives its input in an iterative manner, and must make irrevocable decisions after receiving each part of the input. An online algorithm is evaluated based on the ratio between its cost and the optimal offline cost, that is, the cost which could have been achieved had the entire input sequence been known in advance. The worst-case bound on this ratio is called the *competitive ratio* of the algorithm. Competitive analysis of online algorithms is a very active area of research and the last twenty five years have witnessed many exciting new results. For a broad study of online algorithms, see [2, 3].

Online set cover is a classical online problem; elements arrive one by one, specifying which sets they belong to, and each element must be covered upon arrival. Once a set is chosen to the cover it cannot be removed later on, and its cost is accumulated into the final cost. The objective is to minimize the total cost of the chosen sets. A natural extension is online fractional set cover, where sets are associated with fractional values, and for each element the sum of the fractions of the sets containing it is at least 1. Irrevocability is expressed by requiring the fractional values to be monotonically non-decreasing during the online steps.

* This work was partially supported by ISF grant 1585/15 and BSF grant 2014414.

Set cover further generalizes into general *covering* problems. In this setting, an element is not simply contained in a set, but rather a non-negative weight $a_{e,S}$ is associated with each element-set pair. This is actually equivalent to a linear program where the entries of the constraint matrix are all non-negative. The objective function is also non-negative. Covering problems arise naturally in many settings, e.g., graph optimization problems, facility location and paging. The dual problem of covering is *packing* which also captures an important family of combinatorial problems, e.g., flow, routing, matchings, and combinatorial auctions.

The online primal-dual method is a powerful algorithmic technique that has proved to be extremely useful for a wide variety of problems in the area of online algorithms, serving as an important unifying design methodology. In general, suppose that an online problem can be formulated as a linear program, such that requests correspond to linear constraints. The online nature of the problem translates into the constraint matrix being exposed row by row, while the dual program is updated by adding a new dual variable. The idea behind the primal-dual method is to use both primal and dual programs and construct (simultaneously) feasible solutions for both of them, while maintaining some relationship between their corresponding objective functions. Then, using weak duality, it is easy to bound the competitive factor of the primal solution.

For online covering-packing linear programs, the non-negativity of the constraint matrix facilitates the design of elegant online primal-dual algorithms. This approach has been amazingly successful in both unifying previously known results, as well as resolving several important open questions in competitive analysis. This includes, among others, the classic ski rental problem, the online set-cover problem, paging and weighted paging, graph optimization problems, the dynamic TCP-acknowledgement problem, various routing problems, load balancing, machine scheduling, ad auctions, and more. Please refer for more details to a survey on the covering-packing approach to online algorithms [3].

We explore in this work semidefinite programming in the context of online problems. A symmetric matrix $A \in \mathbb{R}^{m \times m}$ is said to be positive semidefinite, i.e., $A \succcurlyeq 0$, if for every vector $v \in \mathbb{R}^m$, it holds that $v^t A v \geq 0$. A semidefinite program has a constraint requiring that a matrix of variables $X = (x_{i,j})$ is positive semidefinite. Such constraints correspond to an unbounded number of linear constraints, since $X \succcurlyeq 0$ is equivalent to a family of linear constraints: $\forall v \in \mathbb{R}^n \; v^t X v = \sum v_i v_j x_{ij} \geq 0$. This greatly extends our ability to express complex problems, since semidefinite programs can be tailored more specifically, thus decreasing integrality gaps of linear relaxations of combinatorial problems and yielding tighter approximation factors. In their seminal work, Goemans and Williamson [4] used a semidefinite programming relaxation for max cut in undirected graphs, obtaining an approximation factor of 0.878, dramatically improving over the straightforward factor of 0.5.

1.1 Results

We study here further extensions of the successful online primal-dual method and linear programming relaxations for online problems to the realm of semidefinite programming. We define the problem of semidefinite covering-packing, which is a semidefinite program in which all coefficients are non-negative. For matrix coefficients, such as those in constraints of the form $A \bullet X \leq c$, the coefficient matrix A is positive-semidefinite. This problem arises in several natural settings, e.g., quantum hypergraph covering, studied by [1], is a semidefinite covering problem with all matrices restricted to the range $[0, I]$. Our work extends semidefinite covering-packing to an online setting. For example, our extension captures the online covering problem as a special case, in which all matrices are diagonal.

We design an online primal-dual algorithm with competitive ratio $O(\log n)$ for the semidefinite covering-packing setting, where n denotes the number of matrices (e.g., corresponding

to sets in the case of set cover). This is called our *general algorithm* (Section 3). We then consider in Section 4 a class of semidefinite covering-packing problems with *box constraints* on the variables. We design a primal-dual algorithm with an improved bound of $O(\log F^*)$, where F^* is a sparsity parameter coinciding with the maximum row-sparsity D of the constraint matrix in the linear case. This bound can be compared to the $O(\log D)$-competitive algorithm known for covering in the linear case [3, 5].

In Section 5 we design an online randomized rounding procedure, which is applicable to both of our algorithms, while adding a factor of $O(R(\log m + \log n))$ to the solution, where m is the dimension of the matrices and R is a bound on the largest eigenvalue of each matrix. Our randomized rounding uses a general matrix Chernoff bound due to Tropp [6], which is based on and improves slightly the novel matrix Chernoff bound developed by Ahlswede and Winter [1]. In a way, our algorithm is an online variation to Wigderson and Xiaos's [7] randomized rounding algorithm for integer semidefinite programming.

Techniques. Our algorithms and proofs strongly utilize the online primal-dual technique, thus taking advantage of semidefinite weak duality and the fact that semidefinite programming is defined over a cone. Additionally, we use the fact that a semidefinite constraint of the form $A \succcurlyeq B$ can be expressed as an infinite number of linear constraints, each corresponding to a projection onto a vector v satisfying $v^T A v \geq v^T B v$.

The heart of our general algorithm is an update step: when considering a primal semidefinite constraint, it is either already satisfied (and then we are done), or there exists a matrix (in fact, many) which is a witness to the violation of the primal constraint, also inducing a linear constraint which is violated. The witness matrix is then used for updating the primal constraint, and for the direction in which we are going to make dual progress. Once the witness matrix is determined, our update rule becomes quite similar to the update rule in the linear case [3]. An important difference though is that, while only a finite number of (linear) constraints needs to be satisfied in the linear case, now we need to satisfy a semidefinite constraint, equivalent to infinitely many linear constraints. We address this issue by over-satisfying constraints, so as to ensure that we make enough progress for our choice of witness matrix, thus bounding the number of steps needed. Without this over-satisfaction, we might only make infinitesimal progress, and it is not clear if the algorithm ever terminates.

For box constraints, we define the *sparsity* of an online semidefinite program, a measure capturing the potential to overshoot when solving a sub-problem of a semidefinite program. While the definition also coincides with the notion of row-sparsity for linear programs, it is not a simple extension, but rather one which is tailored to a very specific observation about our update rule; each subset of the variables defines such a subproblem when those variables are saturated (i.e. their value equals the upper bound). In this situation we want to choose a good progress direction, so that we do not over-satisfy the semidefinite constraint. The progress direction turns out to have the property that, when projecting the problem onto that direction, minimizes the ratio between the uncovered coefficients and the remainder which is left to cover. This proves to be very useful in bounding the primal-dual ratio, and also in allowing the algorithm to make meaningful progress.

2 Our Model and Definitions

Throughout this paper, all matrices are square, symmetric, real, and have dimension m. Given two matrices A, B, their *Frobenius product*, denoted $A \bullet B$, is defined as $A \bullet B = \sum_{i,j} A_{i,j} B_{i,j} = tr\left(A^T B\right)$. This is equivalent to standard inner product if the matrices are

treated as vectors in $\mathbb{R}^{m^2}$. For any vector $v \in R^n$, it is easy to check that the following useful identity holds: $A \bullet \left(vv^T\right) = v^T A v = \sum_{i,j} v_i v_j A_{i,j}$.

The eigenvalues of a matrix A are $\lambda_1(A) \geq \lambda_2(A) \geq \cdots \geq \lambda_m(A)$. For a matrix A and vector v, the *Rayleigh quotient* is defined as $\frac{v^T A v}{v^T v}$ and has the property $\lambda_m(A) \leq \frac{v^T A v}{v^T v} \leq \lambda_1(A)$. A symmetric matrix A is said to be *positive-semidefinite* (p.s.d.), denoted $A \succcurlyeq 0$, if all its eigenvalues are non-negative. This induces a partial order over $\mathbb{R}^{m \times m}$, where $A \succcurlyeq B$ if and only if $A - B \succcurlyeq 0$.

A*semidefinite* minimization program in standard form and its dual program are:

$$\begin{array}{c} \min \sum_{i=1}^n c_i x_i \\ s.t.\ \sum_{i=1}^n A_i x_i \succcurlyeq B \\ \forall i \in [n]\ \ x_i \geq 0 \end{array} \qquad \begin{array}{c} \max B \bullet Y \\ s.t.\ \forall i \in [n]\ A_i \bullet Y \leq c_i \\ Y \succcurlyeq 0 \end{array} .$$

This pair of programs satisfies weak duality, e.g. every feasible solution of the primal problem is an upper bound on the dual problem and vice versa. This can readily be seen as follows. Let x and Y be feasible solutions for the primal and dual problems, respectively. Then

$$\sum_{i=1}^n c_i x_i \geq \sum_{i=1}^n (A_i \bullet Y) x_i = \left(\sum_{i=1}^n A_i x_i\right) \bullet Y \geq B \bullet Y.$$

The first inequality follows since $x_i \geq 0$ for all $i \in [n]$, and $A_i \bullet Y \leq c_i$. The second inequality is due to $\sum_{i=1}^n A_i x_i \succcurlyeq B$ and $Y \succcurlyeq 0$.

We sometimes use the shorthand $\mathcal{A}x$ to denote the matrix $\sum_{i=1}^n A_i x_i$, that is, $\mathcal{A} : \mathbb{R}^n \to \mathbb{R}^{m \times m}$ is a linear function defined as $\mathcal{A}(x) = \sum_{i=1}^n A_i x_i$.

2.1 Our Model

We define a *semidefinite covering problem* as a semidefinite program in which all scalar coefficients are non-negative ($c_i \geq 0$), and all matrix coefficients are positive semidefinite ($A_i \succcurlyeq 0$, $B \succcurlyeq 0$). We can view the matrix B as being "covered", and the matrices A_i as the "covering" matrices. The variable x_i specifies "how much" of A_i is used to cover B.

We further define as *online semidefinite covering problem* as a semidefinite covering problem in which the covered matrix B is revealed in an online fashion, i.e., in each online step t a new matrix B_t is revealed as the matrix which must be covered, and the following relation holds: $B_t \succcurlyeq B_{t-1}$. The irrevocability property is expressed by the requirement that each variable x_i cannot be decreased at any point of time, and is only allowed to increase (or stay unchanged). In each step t the semidefinite constraint $\mathcal{A}x \succcurlyeq B_t$ must be satisfied.

In the dual *online semidefinite packing problem*, the objective function $B \bullet Y$ is revealed online, while the constraints $A_i \bullet Y \leq c_i$ are known in advance. The monotonicity is captured by allowing the variable matrix Y to only increase, e.g. $Y_t \succcurlyeq Y_{t-1}$.

Set Cover and Linear Covering-Packing as a Special Case. When all matrices A_i and B_t are diagonal, the constraint $\sum_{i=1}^n A_i x_i \succcurlyeq B_t$ defines m linear constraints – one for every diagonal element $\sum_{i=1}^n (A_i)_{j,j} x_i \geq (B_t)_{j,j}$. This is because a diagonal matrix D is p.s.d. if and only if all of its diagonal entries are non-negative. Thus, revealing parts of the matrix B is equivalent to revealing new linear constraints. We assume without loss of generality that the linear constraints are revealed one row at a time, meaning that we reduced the primal problem to an instance of online fractional covering. Specifically, if all the diagonal entries are either 0 or 1, the problem then becomes online set cover. In the dual problem, since Y is only multiplied by diagonal matrices, only its diagonal is taking a part in the program, therefore the dual problem also becomes equivalent to online fractional packing.

3 The General Algorithm

In this section we provide an algorithm for the problem of *online semidefinite covering*, as defined in the previous section:

$$\begin{array}{cc} \min \sum_{i=1}^{n} c_i x_i & \max B_t \bullet Y \\ s.t.\ \sum_{i=1}^{n} A_i x_i \succcurlyeq B_t & s.t.\ \forall i \in [n]\ A_i \bullet Y \leq c_i \\ \forall i \in [n]\ \ x_i \geq 0 & Y \succcurlyeq 0 \end{array} .$$

In each online step t a new matrix B_t is revealed, satisfying $B_t \succcurlyeq B_{t-1}$. Our algorithm must then increase the variables x_i, incurring an additional cost of $c_i \Delta x_i$, such that the updated covering constraint $\sum_{i=1}^{n} A_i x_i \succcurlyeq B_t$ is satisfied.

3.1 Intuition

Our algorithm performs a sort of binary search on the optimal value of the primal problem. For a certain guess α, we either find a primal solution whose value does not exceed α, or find a dual solution whose value is at least $\alpha / O(\log n)$. By doubling our guess each time, we are able to mitigate the cost of failed guesses and only lose a factor of 2 in total. Each guess α defines a phase, which may extend over many online steps. During a phase, the primal solution's value is always less than α, and when the phase ends, the dual solution's value is at least $\alpha / O(\log n)$.

We maintain monotonicity within each phase by only increasing the primal and dual variables. Monotonicity is maintained between phases by setting each variable to be the sum (or the maximum) of its values in the previous phases. We note that the optimal value of the primal problem in each online step can only increase (or at least not change).

The idea behind the algorithm's update step is the following. Either the primal constraint $\mathcal{A}x \succcurlyeq B_t$ is already satisfied (and then we are done), or there exists a matrix $V \succcurlyeq 0$ such that $\mathcal{A}x \bullet V < B_t \bullet V$ (by observing that $C \succcurlyeq 0$ if and only if for every matrix $V \succcurlyeq 0$, $C \bullet V \geq 0$). The requirement that $trV = 1$ is a simple scaling constraint, which can be achieved by setting $V' = V / tr(V)$. We can think of the matrix V as a witness to the violation of the primal constraint; it induces a linear constraint $\mathcal{A}x \bullet V \geq B_t \bullet V$ which is violated. We use the witness as the direction in which we make dual progress, and increase the primal according to the relation we wish to maintain between the primal and dual solutions, until the linear constraint $\mathcal{A}x \bullet V \geq B_t \bullet V$ is satisfied. Specifically, the rate at which x_i is increased is proportional to $A_i \bullet V$, which makes sense, since this is the coefficient of x_i in the linear constraint $\mathcal{A}x \bullet V = \sum_{i=1}^{n} A_i \bullet V x_i \geq B_t \bullet V$.

We note that for technical reasons that help simplify the analysis we actually over-satisfy the above linear constraint, i.e. we continue increasing until $\mathcal{A}x \bullet V \geq \mathbf{2} B_t \bullet V$. This is necessary to ensure that we make enough progress for each choice of V, thus bounding the number of steps needed. Without this requirement, we might make infinitesimal progress for each V, and it is not clear if the algorithm will finish its execution, since there are infinitely many choice of V possible. It turns our that this over-satisfaction only contributes a factor of 2 to the competitive ratio.

There are many possible choices of progress direction V. One natural choice is $V = vv^T$, where v is a unit eigenvector corresponding to the smallest eigenvalue of $(\mathcal{A}x - B_t)$. Since $\mathcal{A}x - B_t \not\succcurlyeq 0$, its smallest eigenvalue λ_n is negative, therefore $(\mathcal{A}x - B_t) \bullet V = v^T (\mathcal{A}x - B_t) v = \lambda_n < 0$ as required.

3.2 Algorithm Description

In each phase r we search for a primal solution with cost at most $\alpha(r)$. Whenever the primal cost exceeds $\alpha(r)$, a new phase starts and $\alpha(r+1) \leftarrow 2\alpha(r)$ is doubled. In each new phase we "forget" about the values of the primal and dual variables from the previous phase, but we do account for their cost in the analysis. That is, in each phase r, new variables $x_{r,i}$, Y_r are introduced; however, since the variables are required to be monotonically non-decreasing, each variable x_i is actually set to $\sum_r x_{r,i}$, and the dual variable is also set to $\sum_r Y_r$.

Let OPT_t be the optimal value of the primal problem in phase t. After the first constraint matrix B_1 is introduced, we set the first lower-bound: $\alpha(1) \leftarrow \min_{i=1}^{n} \frac{c_i tr B_1}{tr A_i}$. Note that $\alpha(1) \leq OPT_1 \leq OPT_t$, for all t, because the matrix $Y_0 = \left(\min_{i=1}^{n} \frac{c_i}{tr A_i}\right) I$ is a feasible solution for the dual problem and its cost is $Y_0 \bullet B_1 = \alpha(1)$. In the beginning of each phase, we initialize $Y = 0$, $x_i = \frac{\alpha(r)}{2nc_i}$. Algorithm 2 describes the phase scheme. Algorithm 1 describes the execution during a single online step t, within a single phase r. Algorithm 1 uses an auxiliary variable δ, the measure of progress by which we update all other variables.

Algorithm 1 Primal-Dual Algorithm for step t within phase r

Input: current solutions x, Y, current limit α. Output: updated values for x, Y.
While $\mathcal{A}x \not\succcurlyeq B_t$:

1. Let V be a density matrix ($V \succcurlyeq 0$, $trV = 1$) such that $\mathcal{A}x \bullet V < B_t \bullet V$.
2. While $\mathcal{A}x \bullet V < 2B_t \bullet V$ and $\sum_{i=1}^{n} c_i x_i < \alpha$:
   a. Set $\delta = 0$ initially, and increase it in a continuous manner.
   b. Increase Y continuously by adding $V\delta$ to it.
   c. Increase x continuously by setting:

$$x_i = \frac{\alpha}{2nc_i} \exp\left(\frac{\log 2n}{c_i} A_i \bullet Y\right).$$

Algorithm 2 Phase Scheme

Initialize $r = 1$, $\alpha(0) = \min_{i=1}^{n} \frac{c_i tr B_1}{2tr A_i}$.
For $t = 1, 2, \ldots$:

1. Let $\alpha(r) = 2\alpha(r-1)$, $Y_r = 0$, $x_{r,i} = \frac{\alpha(r)}{2nc_i}$ for $i = 1, \ldots, n$.
2. Run Algorithm 1 on x_r, Y_r, $\alpha(r)$.
3. If $\sum c_i x_{r,i} \geq \alpha(r)$, then update $r \leftarrow r+1$ and go to step 1.
4. Return solutions $\sum_r x_{r,i}$, $\sum_r Y_r$.

▶ **Theorem 1.**

- *The scheme generates a feasible primal solution with a competitive ratio of* $O(\log n)$.
- *The scheme generates a dual solution with competitive ratio of* $O(\log n)$*, and each constraint is violated by a factor of at most* $O\left(\log\log n + \log \frac{OPT_t}{\alpha(1)}\right)$.
- *The scheme terminates, and its runtime is* $O\left(n\left(\log\log n + \log \frac{OPT_t}{\alpha(1)}\right)\right)$

Before we prove the theorem, we establish some assisting claims. Let $X(r)$ and $Y(r)$ be the values of the primal and dual solutions, respectively, generated in the r-th phase. We say that the r-th phase is *finished* if the condition in step 3 of Algorithm 2 holds, i.e. $\sum c_i x_{r,i} \geq \alpha(r)$.

▶ **Lemma 2.** *For each* finished *phase* r, $Y(r) \geq \alpha(r)/4\log 2n$.

Proof. In the beginning of phase r, we have $x_i = \frac{\alpha(r)}{2nc_i}$, therefore $X(r) = \frac{\alpha(r)}{2}$. The dual cost is zero. When the phase ends, $X(r) \geq \alpha(r)$. The proof is completed by showing that at every point in Algorithm 1, we have $\frac{\partial X(r)}{\partial \delta} \leq 2\log 2n \frac{\partial Y(r)}{\partial \delta}$. To show this, we have

$$\begin{aligned}
\frac{\partial X(r)}{\partial \delta} &= \frac{\partial}{\partial \delta}\left(\sum_{i=1}^{n} c_i x_i\right) = \sum_{i=1}^{n} c_i \frac{\partial x_i}{\partial \delta} = \sum_{i=1}^{n} c_i \frac{\partial}{\partial \delta}\left(\frac{\alpha(r)}{2nc_i}\exp\left(\frac{\log 2n}{c_i} A_i \bullet (Y+\delta V)\right)\right) \\
&= \sum_{i=1}^{n} c_i \frac{\log 2n}{c_i}(A_i \bullet V)\frac{\alpha(r)}{2nc_i}\exp\left(\frac{\log 2n}{c_i} A_i \bullet (Y+\delta V)\right) \\
&= \sum_{i=1}^{n} \log 2n\,(A_i \bullet V)\,x_i = \log 2n\left(\sum_{i=1}^{n} A_i x_i\right) \bullet V \\
&= \log 2n \mathcal{A}x \bullet V \leq 2\log 2n B_t \bullet V = 2\log 2n \frac{\partial Y(r)}{\partial \delta}.
\end{aligned}$$

The last inequality holds since we only increase δ while $\mathcal{A}x \bullet V < 2B_t \bullet V$. ◀

▶ **Lemma 3.** *The dual solution generated during each phase is feasible.*

Proof. Consider the i-th dual constraint of the form $A_i \bullet Y \leq c_i$. Since we are in the r-th phase, the current primal solution's value is at most $\alpha(r)$, therefore the value of x_i can be at most $\frac{\alpha(r)}{c_i}$. Thus:

$$\frac{\alpha(r)}{2nc_i}\exp\left(\frac{\log 2n}{c_i} A_i \bullet Y\right) = x_i \leq \frac{\alpha(r)}{c_i}.$$

Simplifying, we get that $A_i \bullet Y \leq c_i$. The final dual constraint $Y \succcurlyeq 0$ is satisfied since Y is the sum of p.s.d matrices δV. ◀

▶ **Lemma 4.** *The total cost of the primal solutions generated from the first phase until the r-th phase is less than $2\alpha(r)$.*

Proof. We bound the total cost paid by the online algorithm. The total primal cost in the r-th phase is at most $\alpha(r)$. Since the ratio between $\alpha(k)$ and $\alpha(k-1)$ is 2, we get that the total cost until the r-th phase is at most $\sum_{k=1}^{r}\alpha(k) = \alpha(r)\sum_{k=1}^{r}\frac{1}{2^{k-1}} \leq 2\alpha(r)$. ◀

▶ **Lemma 5.** *If the algorithm stops during a certain phase, then x is feasible.*

Proof. The algorithm stops only when $\mathcal{A}x \succcurlyeq B_t$. Also, $x \geq 0$ throughout the entire run of the algorithm, since we only increase each x_i from an initially positive value $\frac{\alpha(r)}{2nc_i} > 0$. ◀

▶ **Lemma 6.** *The number of iterations (= choices of V) in each phase is at most $n+1$.*

Proof. Consider any iteration in Algorithm 1 for which the while loop terminates because $\mathcal{A}x \bullet V \geq 2B_t \bullet V$. In the beginning of the while loop, we have $\mathcal{A}x \bullet V < B_t \bullet V$. This implies that at least one x_i was doubled during the iteration. Now, $x_i \geq \frac{\alpha(r)}{2nc_i}$, therefore each iteration increases the primal value by at least $c_i x_i \geq \frac{\alpha(r)}{2n}$. Since the primal value is $\frac{\alpha(r)}{2}$ in the beginning of the phase, after n such choices of V it must reach $\alpha(r)$ and the phase will be finished. Accounting for the possibly one extra iteration when the while loop terminates because the condition $\sum_i c_i x_i \geq \alpha$ is satisfied, the number of iterations is bounded by $n+1$. ◀

▶ **Lemma 7.** *The number of phases reached by step t is bounded by* $O\left(\log\log n + \log\frac{OPT_t}{\alpha(1)}\right)$.

Proof. Let $\mathcal{R} > 1$ be the current phase. Then

$$\alpha(1)\,2^{\mathcal{R}-2} = \alpha(\mathcal{R}-1) \le 4\log(2n)\,Y(\mathcal{R}-1) \le 4\log(2n)\,OPT_t$$

$$\Rightarrow \mathcal{R} \le 2 + \log\left(2\log(2n)\,\frac{OPT_t}{\alpha(1)}\right) = O\left(\log\log n + \log\frac{OPT_t}{\alpha(1)}\right).$$

The first inequality above follows by Lemma 2, and the second because $Y(\mathcal{R}-1)$ is a feasible solution by Lemma 3. ◀

Proof of Theorem 1. Suppose the online scheme terminates within $\mathcal{R}$ phases. Note that if $\mathcal{R} = 1$ then since $X(1) \le \alpha(1) \le OPT_1$ and x is feasible (from Lemma 5)), we must have exactly reached the end of the phase, i.e. $X = \alpha(1) = OPT_1$, $Y \ge \frac{OPT_1}{\log 2n}$, and Y is also feasible (from Lemma 3). From now on assume $\mathcal{R} > 1$.

- By Lemma 5, the primal solution is feasible. The total primal cost is bounded by:

$$\text{total primal cost} \ \le 2\alpha(\mathcal{R}) = 4\alpha(\mathcal{R}-1) \le 16\log 2nY(\mathcal{R}-1) \le 16\log(2n)\,OPT_t.$$

  The first inequality is by Lemma 4, the second by Lemma 2, and the last one by Lemma 3.
- By Lemma 3, the dual solution in each phase is feasible. Summing up over $\mathcal{R}$ phases and using the bound for $\mathcal{R}$ from Lemma 7, we get that the dual solution is violated up to $\mathcal{R} \le O\left(\log\log n + \log\frac{OPT_t}{\alpha(1)}\right)$. The total dual cost is bounded by:

$$\text{total dual cost} = \sum_{r=1}^{\mathcal{R}} Y(r) \ge Y(\mathcal{R}-1) \ge \frac{\alpha(\mathcal{R}-1)}{4\log 2n} = \frac{\alpha(\mathcal{R})}{8\log 2n} \ge \frac{X(\mathcal{R})}{8\log 2n} \ge \frac{OPT_t}{8\log 2n}.$$

  The second inequality is by Lemma 2 and the last one by Lemma 5.
- The runtime and termination of the scheme follow immediately from Lemmas 6 and 7.

◀

4 Box Constraints

In this section we introduce box constraints to our program, i.e. every variable x_i is now limited to a bounded range $0 \le x_i \le u_i$. Surprisingly, these bounds allows us to achieve approximation factors which do not depend on n, the number of matrices in the primal problem, but rather on a natural property of the program which we call the *sparsity*. Without loss of generality, we can assume $u_i = 1$ by replacing A_i with $A'_i = u_i A_i$, and c_i with $c'_i = u_i c_i$. We note that this assumption does alter the sparsity of the program. Our primal-dual problems with box constraints are the following:

$$\begin{array}{cc} \min \sum_{i=1}^n c_i x_i & \max B_t \bullet Y - \sum_{i=1}^n z_i \\ s.t.\ \sum_{i=1}^n A_i x_i \succcurlyeq B_t & s.t.\ \forall i \in [n]\ A_i \bullet Y - z_i \le c_i \\ \forall i \in [n]\ \ 0 \le x_i \le 1 & Y \succcurlyeq 0,\ z \ge 0 \end{array}.$$

4.1 Sparsity

The sparsity of a semidefinite program is defined as:

$$F^* = \max_t \max_{S \subseteq [n]} \min_{V \succcurlyeq 0:\ \left(B_t - \sum_{i \in S} A_i\right) \bullet V > 0} \frac{\sum_{i \notin S} A_i \bullet V}{\left(B_t - \sum_{i \in S} A_i\right) \bullet V}.$$

This quantity can be viewed as follows. If we saturate all variables in S, then $\sum_{i\in S} A_i$ is the coverage we get from the saturated variables, and $B_t - \sum_{i\in S} A_i$ is the remainder that we still need to cover. $\sum_{i\notin S} A_i$ is the total coverage potential of the unsaturated variables. We project both of these matrices, $\sum_{i\notin S} A_i$ and $B_t - \sum_{i\in S} A_i$, onto some witness V, and take the ratio $\frac{\sum_{i\notin S} A_i \bullet V}{\left(B_t - \sum_{i\in S} A_i\right)\bullet V}$. This is the ratio by which we "over-cover" the remainder $B_t - \sum_{i\in S} A_i$ if we use up all non-saturated variables. It turns out that $F^* \geq 1$. We actually use a relaxed version of F^* which may yield an even better result:

$$F = \max_{t,S \text{ occuring in the algorithm}} \min_{V\succcurlyeq 0:\ \left(B_t - \sum_{i\in S} A_i\right)\bullet V > 0} \frac{\sum_{i\notin S} A_i \bullet V}{\left(B_t - \sum_{i\in S} A_i\right)\bullet V}.$$

Here, the maximum is taken only over each online step t and the corresponding set of saturated variables S at step t. For fixed t, S, we can compute the minimum value of this ratio (and a minimizer V) very efficiently, see Lemma 8 below for details. Although we show that $F \leq F^*$, we still phrase our results using F^*, since F^* only depends on the given program, while F also depends on the specific choices made by the algorithm. We note that it is not necessarily the case that $F^* \leq n$, thus it would only make sense to use this parameter if the specifics of the problem can guarantee a better bound on F^* than n.

To further illustrate the meaning of F^*, it is useful to consider the set cover problem. As discussed earlier, set cover is obtained when all matrices A_i and B_t are diagonal, and having only entries from $\{0, 1\}$ on the diagonal. To analyze F^* in this case, it is sufficient to observe witness matrices V of the form $E_{j,j}$ (any other witness is simply a convex combination of these). Then $\sum_{i\notin S} A_i \bullet V$ and $\sum_{i\in S} A_i \bullet V$ are equal to the number of non-saturated and saturated sets which contain element j respectively; and $B_t \bullet V$ is 1 if element j needs to be covered and 0 otherwise. Clearly the only way in which $\left(B_t - \sum_{i\in S} A_i\right) \bullet V > 0$, is if element j needs to be covered and none of the sets containing it are saturated. Then, the expression $\frac{\sum_{i\notin S} A_i \bullet V}{\left(B_t - \sum_{i\in S} A_i\right)\bullet V}$ is exactly the number of sets containing j. Thus, F^* in this case is exactly the maximum number of sets that an element is contained in (taken over the elements which the algorithm is required to cover). This matches the notion of *row sparsity* of a linear program, and clearly in this case $F^* \leq n$.

▶ **Lemma 8.** *Let N be a positive semidefinite matrix and D be a symmetric matrix. Then the minimum value of the ratio $\frac{N\bullet V}{D\bullet V}$ such that $V \succeq 0$ and $D \bullet V > 0$ is equal to the smallest non-negative root of the polynomial equation $\det(N - \lambda D) = 0$.*

Proof. Let V be an optimal solution. Since $V \succeq 0$, we can express it as $V = \sum_\ell v_\ell v_\ell^T$ for some vectors v_ℓ. Since $N \succeq 0$, we have $N \bullet v_\ell v_\ell^T \geq 0$. Then we claim that $D \bullet v_\ell v_\ell^T \geq 0$; otherwise, we could drop $v_\ell v_\ell^T$ from the sum forming V and decrease the ratio. Next, note that $\frac{N\bullet V}{D\bullet V} = \frac{\sum_\ell N\bullet v_\ell v_\ell^T}{\sum_\ell D\bullet v_\ell v_\ell^T} \geq \min_\ell \left\{\frac{N\bullet v_\ell v_\ell^T}{D\bullet v_\ell v_\ell^T}\right\}$, and so we conclude that there is a rank 1 minimizer. Let this minimizer be $V = vv^T$ for some vector v. The minimum ratio is then obtained by minimizing $v^T N v$ subject to $v^T D v = 1$. The Karush-Kuhn-Tucker conditions imply that the optimal vector v satisfies the generalized eigenvalue equation, $Nv = \lambda Dv$ for some constant λ (the generalized eigenvalue). Note that $\frac{N\bullet vv^T}{D\bullet vv^T} = \lambda$. Since we require $v^T D v = 1$, and $v^T N v \geq 0$, only non-negative values of λ are admissible. Since the generalized eigenvalues λ are roots of the polynomial equation $\det(N - \lambda D) = 0$, and we conclude that the minimum value of the ratio is the smallest non-negative root of this polynomial equation. ◀

4.2 Algorithm Description

We now describe an algorithm for semidefinite covering-packing with box constraints. We denote by S the set of indices i of *saturated* primal variables x_i, i.e. variables which have reached their upper bound: $S = \{i | x_i = 1\}$. We use the following notations for simplicity: $\mathcal{A}_{\bar{S}} x = \sum_{i \notin S} A_i x_i$, and $\mathcal{A}_S x = \sum_{i \in S} A_i$ (note that in the second notation, we omit the x_i's because they are all saturated, so are equal to 1). We use an auxiliary variable δ, which is the measure of progress by which we update all other variables.

Algorithm 3 Primal-Dual Algorithm For Box Constraints

Initialize $x = 0$, $z = 0$, $Y = 0$.

Upon arrival of a new constraint matrix B_t, while $\mathcal{A}x \not\succcurlyeq B_t$:

1. Let V be a density matrix ($V \succcurlyeq 0$, $trV = 1$) such that $\mathcal{A}x \bullet V < B_t \bullet V$, minimizing the ratio $\frac{\sum_{i \notin S} A_i \bullet V}{B_t \bullet V - \sum_{i \in S} A_i \bullet V}$.
2. Update $F = \max\left\{F, \frac{\sum_{i \notin S} A_i \bullet V}{B_t \bullet V - \sum_{i \in S} A_i \bullet V}\right\}$.
3. While $\mathcal{A}_{\bar{S}} x \bullet V < 2\left[B_t \bullet V - \mathcal{A}_S x \bullet V\right]$:
   a. Set $\delta = 0$ initially, and increase it in a continuous manner.
   b. Increase Y continuously by adding $V\delta$ to it.
   c. For every i such that $x_i = 1$, increase z_i continuously at rate $A_i \bullet V\delta$.
   d. Update x continuously by setting:

$$x_i = \max\left\{x_i, \frac{1}{F}\left(\exp\left(\frac{\log(F+1)}{c_i}\left(A_i \bullet Y - z_i\right)\right) - 1\right)\right\}.$$

▶ **Theorem 9.** *The algorithm produces a feasible primal solution with a competitive ratio of* $O(\log F^*)$*, and a feasible dual solution with a competitive ratio of* $O(\log F^*)$*.*

The proof is omitted due to space constraints.

5 Rounding

In this section, we discuss the integer version of online semidefinite programming, and how our results in the previous sections can be rounded to an integer solution.

5.1 Changing the Framework

An *integer semidefinite program* is the following:

$$\begin{array}{c} \min \sum_{i=1}^{n} c_i \hat{x}_i \\ s.t. \ \sum_{i=1}^{n} A_i \hat{x}_i \succcurlyeq I \\ \forall i \in [n] \ \ \hat{x}_i \in \mathbb{N}^n \end{array}.$$

Note the different notation for integer variables $\hat{x}_i$ as opposed to real-valued variables x_i of the previous sections. Also note that we now restrict the covered matrix to be the identity matrix I. This is a necessity for our rounding scheme. To justify why this is not any different from the general setting, we invoke some linear algebra. First, assume that B is of full rank (and, of course, p.s.d.). Then B is congruent to I, by some invertible matrix X such that

$X^T BX = I$. Now $\sum_{i=1}^n A_i \hat{x}_i \succcurlyeq B$ if and only if $\sum_{i=1}^n X^T A_i X \hat{x}_i \succcurlyeq X^T BX = I$. We can change the matrices $A'_i = X^T A_i X$ so as to arrive at the formulation above.

However, restricting B to be of full rank throughout the entire online algorithm does not make much sense. To overcome this, we change the framework to allow the dimension m of the matrices to increase over time. Let m_t be the dimension of the online problem at step t. Then, in step $t+1$, the dimension of the matrices may increase to $m_t + 1$ (or may stay m_t). The covering matrices $A_i^{(t+1)}$ must contain $A_i^{(t)}$ as their top-left submatrix, and they must be p.s.d., i.e. $A_i^{(t+1)} \succcurlyeq 0$. The covered matrix $B^{(t+1)}$ must also be p.s.d., and its top-left submatrix of dimension m_t must be greater than or equal to $B^{(t)}$ in the p.s.d. partial ordering. The *online integer semidefinite program* is therefore:

$$\begin{array}{c} \min \sum_{i=1}^n c_i \hat{x}_i \\ s.t. \ \sum_{i=1}^n A_i^{(t)} \hat{x}_i \succcurlyeq B^{(t)} \\ \forall i \in [n] \ \ \hat{x}_i \in \mathbb{N}^n \end{array},$$

where $A_i^{(t)}, B^{(t)} \in \mathbb{R}^{m_t \times m_t}$, and $B^{(t)}$ is of full rank for all t.

Under these assumptions, it can be verified that our algorithms from Sections 3 and 4 still work (for real-valued solutions $x_i \in \mathbb{R}^n$). For Section 3, it is important to note that OPT_t indeed increases with t. The proof of the next lemma is omitted due to space constraints.

▶ **Lemma 10.** *With the above assumptions, $OPT_t \leq OPT_{t+1}$.*

Another consideration when discussing rounding is scale – if $A_1 = (M)$ and $B = (1)$ (matrices of dimension 1), then the optimum fractional solution is $x_1 = \frac{1}{M}$ but the optimum integer solution is 1, which costs M times the fractional solution. Therefore we must assume some kind of bound on the covering matrices (we use $\lambda_1(A_i) \leq R$), and we will be paying for this bound in our competitive ratio. As demonstrated by this example, this is unavoidable.

5.2 A Randomized Rounding Algorithm

We use a simple randomized rounding scheme. We run the algorithm described in Section 3[1] to maintain a fractional solution $x^{(t)}$, and we round that solution to an integer solution $\hat{x}^{(t)}$, so that $\mathbb{E}\hat{x}^{(t)} = \rho_t x^{(t)}$, where ρ_t is an approximation factor (see algorithm for exact definition), which we adjust as the dimension of the matrices grows. We then use the feasibility of $x^{(t)}$ to show that w.h.p. $\hat{x}^{(t)}$ is also feasible. The rounding is done as follows: whenever a variable x_i increases from $x_i^{(t-1)}$ to $x_i^{(t)}$, we add 1 to $\hat{x}_i$ with probability $\rho_t x_i^{(t)} - \rho_{t-1} x_i^{(t-1)}$. Since m_t and $x^{(t)}$ can only increase with t, so can ρ_t and thus $\rho_t x_i^{(t)} - \rho_{t-1} x_i^{(t-1)}$ is non-negative. To make sure that this value is always ≤ 1, we can simply break up the increase in x_i into smaller steps. This is expressed formally in algorithm 4.

▶ **Theorem 11.** *The resulting solution's expected cost is at most $O(R(\log m_t + \log n))$ times the cost of the fractional solution.*

The proof is omitted due to space constraints.

[1] If box constraints are given, we can run the algorithm from Section 4 instead. The only modification to the rounding is that each variable is increased at most once. The same analysis holds for both cases.

Algorithm 4 Randomized rounding

- Let $R > 0$ be such that $\lambda_1(A_i) \leq R$ for all i.
- Initialize $\hat{x}^{(0)} = 0$
- For step $t = 1, 2, \ldots$:
  - Let $\rho_t = 1 + \max\{8R(\log 2m_t + \log n), 2\}$.
  - Use Algorithms (2) or (3) to compute the fractional solution $x^{(t)}$.
  - For $i = 1, \ldots, n$:
    - Let $d_{i,t} = \left\lfloor \rho_t x_i^{(t)} - \rho_{t-1} x_i^{(t-1)} \right\rfloor$, and $\delta_{i,t} = \rho_t x_i^{(t)} - \rho_{t-1} x_i^{(t-1)} - d_{i,t}$
    - Let $\hat{x}_{i,1}^{(t)}, \ldots, \hat{x}_{i,d_{i,t}}^{(t)} = 1$.
    - Let $\hat{x}_{i,d_{i,t}+1}^{(t)} = \begin{cases} 1 & \text{with probability } \delta_{i,t} \\ 0 & \text{otherwise} \end{cases}$.
    - return $\hat{x}_i^{(t)} = \hat{x}_i^{(t-1)} + \hat{x}_{i,1}^{(t)} + \cdots + \hat{x}_{i,d_{i,t}+1}^{(t)}$.

5.3 An Application – Quantum Hypergraph Covering

Quantum hypergraphs originate from the field of quantum information theory. A hypergraph is a pair (V, E) where $E \subseteq 2^V$. The hypergraph covering problem (finding a collection of edges covering all vertices) is equivalent to set cover. Let each edge $e \in E$ be represented by a $\{0,1\}$ diagonal matrix A_e, where the i-th diagonal entry is 1 if and only if the i-th element in V belongs to e. The hypergraph covering problem is to find a collection of edges such that $\sum_{e \in cover} A_e \succcurlyeq I$. A *quantum hypergraph* is a pair (V, E), where each $e \in E$ corresponds to a symmetric matrix A_e of dimension $|V|$, such that $0 \preccurlyeq A_e \preccurlyeq I$. The difference is that A_e need not be diagonal or only have $\{0,1\}$ eigenvalues. The *quantum hypergraph covering* problem is thus the problem of finding a minimum collection of edges in E that satisfy $\sum_{e \in cover} A_e \succcurlyeq I$. The *fractional quantum hypergraph covering* problem is the problem of assigning non-negative weights x_e to each $e \in E$ such that $\sum_{e \in E} x_e A_e \succcurlyeq I_{|V|}$, while minimizing $\sum_{e \in E} x_e$.

Ahlswede and Winter [1] used a novel approach to develop a matrix-valued generalization of the Chernoff inequality. They applied this inequality in the analysis of their randomized rounding scheme, which takes a fractional quantum hypergraph cover and rounds it to a quantum hypergraph cover. They showed that this rounding finds a cover which is at most $O(\log m)$ times larger than the minimum cover, where m is the dimension of the matrices involved. Wigderson and Xiao [7] derandomized this result to provide a deterministic algorithm for quantum hypergraph covering. Their result also produces an $O(\log m)$-approximation cover. In addition to the quantum hypergraph covering problem, Wigderson and Xiao also provided a more general algorithm for integer semidefinite programming.

It is easy to see that the fractional quantum hypergraph covering problem is exactly the semidefinite covering problem, with the added requirement that all the covering matrices A_e fall in the range $[0, I]$, and the covered matrix is the identity matrix $B = I$. When translating this into an online setting, the most natural approach is to look at the setting presented in Section 5, where the dimension of the matrices can increase but the covered matrix must have full rank. The *online quantum hypergraph covering problem* is thus the problem of maintaining a monotonically increasing cover while the edges are revealed one dimension at a time. In this setting, combining algorithms 2 and 4 gives an $O(\log n \log m)$-competitive solution to online quantum hypergraph programming ($R = 1$ since $0 \preccurlyeq A_e \preccurlyeq I$).

References

1 Rudolf Ahlswede and Andreas Winter. Strong converse for identification via quantum channels. *Information Theory, IEEE Transactions on*, 48(3):569–579, 2002.

2 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

3 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Mathematics of Operations Research*, 34(2):270–286, 2009.

4 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.

5 Anupam Gupta and Viswanath Nagarajan. Approximating sparse covering integer programs online. In *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 436–448, 2012.

6 Joel A Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012.

7 Avi Wigderson and David Xiao. Derandomizing the AW matrix-valued Chernoff bound using pessimistic estimators and applications, 2006.