

Directed Studies 2.

Matrix Multiply Weights Algorithm and its connections to quantum computing

Attila Monos
Advisor: András Gilyén

I. INTRODUCTION

A generalization of Linear programs (LPs) is Semidefinite programs (SDPs), which works with matrices as variables instead of vectors. As expected, this poses a challenge of matrix operations being more complex both to reason about and to implement.

However the Matrix Multiplicative Weights Algorithm – the main object of my research – offers a meta-algorithm to solve many problems either faster or with more precision using SDPs instead of LPs.

During my studies, first I familiarized myself with the Multiplicative Weights Algorithm on vectors through an approach by game theory. Then I've broadened my understanding to matrices in the same field; after that, I've learned what SDPs are and examined the meta-algorithm of solving SDPs by the Matrix Multiplicative Weights Algorithm.

II. MULTIPLICATIVE WEIGHTS ALGORITHM

A. Weighted Majority Algorithm

Consider the following problem (*Prediction from Expert Advice*)¹: we have to make decisions daily whether to invest in a stock or not on that day. We invest in the stock if the price is predicted to go up, and we don't if it's predicted to go down. To make this decision, we try to predict whether the price will go up, and make our decision based on our prediction. If our prediction is wrong, we lose a unit of our invested money that day, if it's correct, we don't. Our goal is to find an investment strategy that minimizes our cumulative losses.

The movements of the stock are unknown to us, so we might have to predict blindly. However, we are allowed to watch the predictions of n experts. The decision making progress of these experts is unknown as well, we'll only know who the best expert was at the end of our decision making sequence. But we have to make the decision before knowing that. Thus we'll change our goal to limit our cumulative losses to be roughly the same as of the best of the given experts.

The main idea of the algorithm is to somehow take note of the effectiveness of the experts and give more weight to those experts who make less bad decisions. This way, we'll

compute our own prediction based on the opinion of the weighted majority of the experts (thus avoiding the scenario of a majority of experts making consistently bad decisions and us following those decisions). Notice that while any good

Algorithm 1 Weighted Majority Algorithm²

- 1: Fix an $\eta \leq \frac{1}{2}$. With each expert i associate the weight $w_i^{(1)} = 1$.
- 2: **for** $t = 1, 2, \dots, T$ **do**
- 3: Make the prediction of stock going up or down based on the weights $w_1^{(t)}, \dots, w_n^{(t)}$: choose the one with higher weights. In the case of a tie, choose arbitrarily.
- 4: For every expert i making a wrong prediction, decrease the weight by the following update rule:

$$w_i^{(t+1)} = (1 - \eta)w_i^{(t)}$$

- 5: **end for**
-

decision made by an expert keeps said expert's weight intact, any bad decision made decreases the weight we put to said expert's word – thus the better the expert, the more we listen to it. The following theorem³ highlights the power of this algorithm:

Theorem 2.1: Let $m_i^{(t)}$ be the number of mistakes made by expert i and $M^{(T)}$ be the number of mistakes made by the Weighted Majority Algorithm. Then for every i the following holds:

$$M^{(T)} \leq 2(1 + \eta)m_i^{(T)} + \frac{2 \ln n}{\eta}$$

Note that this stands for the best expert as well, meaning that we can give an upper bound for the amount of mistakes we made based upon the amount of mistakes the best expert has made.

B. Multiply Weights Algorithm

Generalizing the setting of the previous subsection, the problem is the following: each round, we have to pick a decision from a set of n decisions. In each round, each decision incurs a cost which is determined by an outside party. These costs are revealed to us after choosing our decision, and we incur the cost of our chosen decision. Our goal is to minimize to cumulative cost of our decisions. In the previous problem,

$n = 2$, and the cost was 1 if we predicted wrongly, 0 if we predicted right.

Initially, since we have no knowledge about any costs, we – ignorantly – pick a decision at random. After we incur the costs, we use our knowledge to change the probabilities of the decisions. By this move, we lower the entropy, and ensure that at least the expected cost of our decision is lower.

Let's note the distribution in round t by $\mathbf{p}^{(t)}$, the cost of the decisions $\mathbf{m}^{(t)}$. We'll assume that the elements of the cost vector lie in the interval $[-1; 1]$, but make no other assumptions about the cost vector. In this case, the expected cost in round t is $\mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$, and the total expected cost over all rounds is

$$\sum_{t=1}^T \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$$

The goal is an algorithm that the total expected cost is not too much worse than the cost of the best decisions in hindsight. In other words, we want to minimize $\sum_{t=1}^T \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)}$. It is

Algorithm 2 Multiply Weights algorithm⁴

- 1: Fix an $\eta \leq \frac{1}{2}$. With each decision i associate the weight $w_i = 1$
- 2: **for** $t = 1, 2, \dots, T$ **do**
- 3: Choose decision i with probability proportional to its weight $w_i^{(t)}$.
- 4: Observe the costs of decisions $\mathbf{m}^{(t)}$.
- 5: Penalize the costly decisions: for every decision i set

$$w_i^{(t+1)} = w_i^{(t)}(1 - \eta m_i^{(t)})$$

6: **end for**

useful to note that for negative costs (which means that we actually gain something by making a decisions) increase the weight of said decision.

The probability proportional to the weight $w_i^{(t)}$ is the following:

$$p_i^{(t)} = \frac{w_i^{(t)}}{\sum_{i=1}^n w_i^{(t)}}$$

A similar theorem⁵ is true for this more general meta-algorithm too:

Theorem 2.2: Assume that all costs $m_i^{(t)} \in [-1; 1]$ and $\eta \leq \frac{1}{2}$. Then the Multiply Weight algorithm guarantees that after T rounds, for any decision i , we have:

$$\sum_{t=1}^T \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \leq \sum_{t=1}^T m_i^{(t)} + \eta \sum_{t=1}^T |m_i^{(t)}| + \frac{\ln n}{\eta},$$

By taking a convex combination of the inequalities for all decisions i with the distribution p , we get the following corollary⁶:

Corollary 2.1: The Multiplicative Weights algorithm also guarantees that after T rounds, for any distribution \mathbf{p} on the

decisions,

$$\sum_{t=1}^T \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \leq \sum_{t=1}^T (\mathbf{m}^{(t)} + \eta |\mathbf{m}^{(t)}|) \cdot \mathbf{p} + \frac{\ln n}{\eta},$$

where $|\mathbf{m}^{(t)}|$ is taken coordinate-wise.

Another popular update rule is to use an exponential factor to update the weights. For example, the Hedge algorithm of Freund and Schapire uses this factor:

$$w_i^{(t+1)} = w_i^{(t)} \cdot \exp(-\eta m_i^{(t)})$$

This leads to a slightly different upper bound⁷:

Theorem 2.3: Assume that all costs $m_i^{(t)} \in [-1; 1]$ and $\eta \leq 1$. Then the Hedge algorithm guarantees that after T rounds, for any decision i , we have an upper bound of

$$\sum_{t=1}^T \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \leq \sum_{t=1}^T m_i^{(t)} + \eta \sum_{t=1}^T (\mathbf{m}^{(t)})^2 \cdot \mathbf{p}^{(t)} + \frac{\ln n}{\eta},$$

where $(\mathbf{m}^{(t)})^2$ is taken coordinate-wise.

Another version of the Multiplicative Weights Update algorithm uses the Kullback-Leibler divergence (also known as relative entropy):

$$\mathbf{RE}(\mathbf{p} \parallel \mathbf{q}) = \sum_i p_i \ln \frac{p_i}{q_i}$$

where the term in the sum is 0 if $p_i = 0$ and infinite if $p_i \neq 0$, $q_i = 0$.

In this version, we assume that the distribution \mathbf{p} comes from a fixed convex set \mathcal{P} of distributions. The previous algorithms are cases where \mathcal{P} is the set of all distributions over the set of decisions.

Now we will compare the cost of our decisions to the total cost of the best fixed distribution from \mathcal{P} . This approach will result in a tighter bound⁹:

Theorem 2.4: Assume that all costs $m_i^{(t)} \in [-1; 1]$ and $\eta \leq \frac{1}{2}$. Then the Multiply Weights algorithm with Restricted Distributions guarantees that after T rounds, for any $\mathbf{p} \in \mathcal{P}$, we have:

$$\sum_{t=1}^T \mathbf{m}^{(t)} \cdot \mathbf{p}^{(t)} \leq \sum_{t=1}^T (\mathbf{m}^{(t)} + \eta |\mathbf{m}^{(t)}|) \cdot \mathbf{p} + \frac{\mathbf{RE}(\mathbf{p} \parallel \mathbf{p}^{(1)})}{\eta},$$

where $|\mathbf{m}^{(t)}|$ is taken coordinate-wise.

It depends on the problem which version is the best to use. The algorithm is the following if we restrict the distributions to \mathcal{P} :

Algorithm 3 Multiply Weights Update algorithm with Restricted Distributions⁸

- 1: Fix an $\eta \leq \frac{1}{2}$. Set $\mathbf{p}^{(1)}$ to be an arbitrary distribution in \mathcal{P} initialized to 1.
- 2: **for** $t = 1, 2, \dots, T$ **do**
- 3: Choose decision i by sampling from $\mathbf{p}^{(t)}$.
- 4: Observe the costs of decisions $\mathbf{m}^{(t)}$.
- 5: Compute the probability vector $\hat{\mathbf{p}}^{(t+1)}$ by the usual multiplicative rule: for every expert i , let

$$\hat{p}_i^{(t+1)} = p_i \frac{1 - \eta m_i^{(t)}}{\Phi^{(t)}},$$

where $\Phi^{(t)}$ is the normalization factor that makes $\hat{\mathbf{p}}^{(t+1)}$ a distribution.

- 6: Set $\mathbf{p}^{(t+1)}$ to be the relative entropy projection of $\hat{\mathbf{p}}^{(t)}$ on the set \mathcal{P} :

$$\mathbf{p}^{(t+1)} = \arg \min_{\mathbf{p} \in \mathcal{P}} \mathbf{RE}(\mathbf{p} \parallel \hat{\mathbf{p}}^{(t)})$$

- 7: **end for**
-

C. A MW algorithm solving a LP

Consider a matrix game where the two players (Alice and Bob) can choose from moves labeled by $[n]$ and $[m]$ respectively. If Alice plays $i \in [n]$ and Bob plays $j \in [m]$ then Alice gets a payoff $A_{ij} \in [-1; 1]$ and Bob gets a payoff $-A_{ij}$. Their individual goal is to get the highest payoff possible. The payoff can be written in matrix form $A \in [-1; 1]^{n \times m}$ – the matrix game is symmetric if $m = n$ and $A = -A^T$.

If either Alice or Bob would always play the same move, or play in a predictable way, then for most games the other player could easily win. Thus the strategy will be randomized. Let Δ^n be the n -dimension probability simplex, and fix the randomized strategies $x \in \Delta^n$, $y \in \Delta^m$ for Alice and Bob. In this case, the expected payoff for Alice is $x^T A y$. Bob's goal is to minimize Alice's expected payoff, and by doing that, maximize his own expected payoff. He assumes that Alice plays the best strategy x and optimizes his strategy y for that:

$$\min_{y \in \Delta^m} \max_{x \in \Delta^n} x^T A y$$

This can be written as a LP by noting that $x \mapsto x^T A y$ is a linear function over the probability simplex and is maximized on a vertex:

$$\min_{y \in \Delta^m} \max_{x \in \Delta^n} x^T A y = \min_{y \in \Delta^m} \max_{i \in [n]} e_i^T A y,$$

where e_i is the i th unit vector, and e is the all-one vector. This problem can be written as a LP for which strong duality holds. Thus the primal problem:

$$\begin{aligned} \min \quad & \lambda \\ \text{s.t.} \quad & A y \leq \lambda e \\ & y \in \Delta^m \\ & \lambda \in [-1; 1] \end{aligned}$$

And the dual problem:

$$\begin{aligned} \max \quad & \lambda' \\ \text{s.t.} \quad & A^T x \geq \lambda' e \\ & \lambda' \in \Delta^n \\ & \lambda' \in [-1; 1] \end{aligned}$$

The Multiplicative Weights algorithm for solving this LP is the following: The following statements¹¹ show that in a

Algorithm 4 Modified Grigoriados-Khachiyan algorithm (MGK algorithm)¹⁰

- 1: $x^{(0)} \leftarrow 0 \in \mathbb{R}^n$ and $y^{(0)} \leftarrow 0 \in \mathbb{R}^m$
 - 2: **for** $t = 1, 2, \dots$ **do**
 - 3: $\eta^{(t)} = \frac{1}{2\sqrt{t}}$
 - 4: $u^{(t)} \leftarrow -A^T x^{(t)}$ and $v^{(t)} \leftarrow A y^{(t)}$
 - 5: $P^{(t)} \leftarrow \exp(u^{(t)}) = \exp(-A^T x^{(t)})$ and $Q^{(t)} \leftarrow \exp(v^{(t)}) = \exp(A y^{(t)})$
 - 6: $p^{(t)} \leftarrow \frac{P^{(t)}}{\|P^{(t)}\|_1}$ and $q^{(t)} \leftarrow \frac{Q^{(t)}}{\|Q^{(t)}\|_1}$
 - 7: Sample $a \sim p^{(t)}$ and $b \sim q^{(t)}$
 - 8: $y^{(t+1)} = y^{(t)} + \eta^{(t)} e_a$ and $x^{(t+1)} = x^{(t)} + \eta^{(t)} e_b$.
 - 9: **end for**
-

finite number of steps, this algorithm can give an ε -optimal y strategy for Bob:

Theorem 2.5:

- Let $\delta \in (0, \frac{1}{3})$. With probability at least δ the MGK algorithm produces a sequence of $x^{(t)}$ and $y^{(t)}$ such that for all t the intermediate solutions $\frac{x^{(t)}}{\|x^{(t)}\|_1}$ and $\frac{y^{(t)}}{\|y^{(t)}\|_1}$ are ε -optimal solutions with

$$\varepsilon = \frac{2}{\sqrt{t}} \cdot (3 \ln t + \ln(nm) + \ln\left(\frac{1}{\delta}\right) + 2)$$

- Given a pair of strategies x and y and $k = \mathcal{O}(\frac{1}{\varepsilon^2})$ independent samples i_1, \dots, i_k from x and j_1, \dots, j_k from y , $\frac{1}{k} \sum_{l=1}^k A_{i_l j_l}$ is an ε -approximate estimate of the value of the game corresponding to strategies x, y .
- The MGK algorithm can be implemented to find an ε -optimal pair of solutions with probability at least $1 - \delta$ in $\frac{16}{\varepsilon^2} \ln(\frac{nm}{\delta})$ iterations, using $\mathcal{O}(n + m)$ time per iteration and $n + m$ queries to the entries of A .

This theorem gives us a way of using multiplicative weights to find an approximate solution fairly quickly for an LP.

III. SEMIDEFINITE PROGRAMS

Semidefinite programs are generalizations of LPs which are quite useful in designing approximation algorithms for NP-hard problems such as the MAX CUT, SPARSEST CUT, MIN UNCUT, MIN 2CNF DELETION problems. The main aim of my research was to understand how the Multiply Weights algorithm can be adapted to work with Semidefinite programs.

A. A Semidefinite Program

For symmetric matrices \mathbf{A} and \mathbf{B} , $\text{Tr}(\mathbf{A})$ denotes the trace of \mathbf{A} . The dot product of \mathbf{A} and \mathbf{B} is $\mathbf{A} \bullet \mathbf{B} = \text{Tr}(\mathbf{A}^T \mathbf{B}) = \sum_{ij} A_{ij} B_{ij}$. \mathbf{A} is a positive semidefinite matrix if and only if $\mathbf{A} \bullet \mathbf{B} \leq 0$ for all positive semidefinite \mathbf{B} matrices. $\mathbf{A} \succeq \mathbf{B}$ if $\mathbf{A} - \mathbf{B}$ is positive semidefinite (so $\mathbf{A} - \mathbf{B} \succeq 0$). $\|\mathbf{A}\|$ is the spectral norm: maximum absolute value of eigenvalues of \mathbf{A} . $\|\mathbf{A}\|_F$ is the Frobenius norm: $\|\mathbf{A}\|_F = \sqrt{\sum_{ij} A_{ij}^2} = \sqrt{\mathbf{A} \bullet \mathbf{A}}$.

The exponential of the matrix \mathbf{A} is $\exp(\mathbf{A}) = \sum_{k=0}^{\infty} \frac{\mathbf{A}^k}{k!}$. Since \mathbf{A} is symmetric, $\exp(\mathbf{A}) \succeq 0$. Every positive semidefinite matrix has a Gram factorization: $\mathbf{A} = \mathbf{V}^T \mathbf{V}$. If we denote the vectors of \mathbf{V} with \mathbf{v}_i , then $A_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$. Matrix exponentiation itself can have accuracy issues¹² while computing, but we can approximate the Gram factorization in such a way that the lengths of \mathbf{v}_i and $\mathbf{v}_i - \mathbf{v}_j$ are preserved up to a multiplicative factor $1 + \varepsilon$.

IV. MATRIX MULTIPLICATIVE WEIGHTS ALGORITHM

A. The Matrix Multiplicative Weights (MMW) algorithm

For a symmetric matrix \mathbf{A} , let's denote it's eigenvalues by $\lambda_1(\mathbf{A}) \geq \lambda_2(\mathbf{A}) \geq \dots \geq \lambda_n(\mathbf{A})$.

A generalization of our problem is the following: Alice chooses a unit vector $\mathbf{v} \in \mathbb{S}^{n-1}$. Bob chooses a symmetric loss matrix \mathbf{M} with eigenvalues in $[-1; 1]$ (thus $\|\mathbf{M}\| \leq 1$). Alice pays Bob $\mathbf{v}^T \mathbf{M} \mathbf{v} = \mathbf{M} \bullet \mathbf{v} \mathbf{v}^T$. Alice can choose \mathbf{v} from a distribution \mathcal{P} over \mathbb{S}^{n-1} . The point of interest is the expected loss of Alice:

$$\mathbb{E}_{\mathcal{P}}(\mathbf{v}^T \mathbf{M} \mathbf{v}) = \mathbf{M} \bullet \mathbb{E}_{\mathcal{P}}(\mathbf{v} \mathbf{v}^T)$$

The matrix $\mathbf{P} = \mathbb{E}_{\mathcal{P}}(\mathbf{v} \mathbf{v}^T)$ is a density matrix: positive semidefinite with trace 1. These matrices appear in quantum computation – this is the reason that the MMW algorithm is useful for quantum computing.

In each round, Alice reacts to a loss matrix \mathbf{M} picked by Bob. Alice does this by choosing a density matrix $\mathbf{P}^{(t)}$ and observing the loss matrix $\mathbf{M}^{(t)}$. The loss matrix can depend on the matrices $\mathbf{P}^{(\tau)}$ and $\mathbf{M}^{(\tau)}$ for $\tau = 1, 2, \dots, t-1$ but is independent of $\mathbf{P}^{(t)}$. After T rounds have passed, the best fixed vector for Alice in hindsight is the unit vector \mathbf{v} for which $\sum_{t=1}^T \mathbf{v}^T \mathbf{M}^{(t)} \mathbf{v}$ is minimal.

This happens when \mathbf{v} is the unit eigenvector of $\sum_{t=1}^T \mathbf{M}^{(t)}$ corresponding to its smallest eigenvalue. The goal of the MMW algorithm is that the total expected loss over these T rounds is not much more than $\lambda_n(\sum_{t=1}^T \mathbf{M}^{(t)})$. A theorem¹⁵ similar to the previous ones I've mentioned hold here as well:

Theorem 4.1: For any sequence of loss matrices $\mathbf{M}^{(1)}, \mathbf{M}^{(2)}, \dots, \mathbf{M}^{(T)}$, the MMW algorithm generates

Algorithm 5 Matrix Multiplicative Weights (MMW) Algorithm^{13,14}

- 1: Fix $\eta \leq 1$, number of rounds T .
- 2: Set $\mathbf{W}^{(1)} = \mathbf{I}$.
- 3: **for** $t = 1, 2, \dots, T$ **do**
- 4: Use the density matrix $\mathbf{P}^{(t)} = \frac{\mathbf{W}^{(t)}}{\text{Tr}(\mathbf{W}^{(t)})}$
- 5: Obtain loss matrix $\mathbf{M}^{(t)}$ from Bob
- 6: Update the weight matrix as follows:

$$\mathbf{W}^{(t+1)} = \exp\left(-\eta \sum_{\tau=1}^t \mathbf{M}^{(\tau)}\right)$$

- 7: **end for**
-

density matrices $\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \dots, \mathbf{P}^{(t)}$ such that

$$\sum_{t=1}^T \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \leq \lambda_n \left(\sum_{t=1}^T \mathbf{M}^{(t)} \right) + \eta \sum_{t=1}^T (\mathbf{M}^{(t)})^2 \bullet \mathbf{P}^{(t)} + \frac{\ln n}{\eta}.$$

This theorem has the following corollary¹⁶ due to the fact that $\|\mathbf{M}^{(t)}\| \leq 1$:

Corollary 4.1: For any sequence of loss matrices $\mathbf{M}^{(1)}, \mathbf{M}^{(2)}, \dots, \mathbf{M}^{(T)}$, the MMW algorithm generates density matrices $\mathbf{P}^{(1)}, \mathbf{P}^{(2)}, \dots, \mathbf{P}^{(t)}$ such that

$$\sum_{t=1}^T \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \leq \lambda_n \left(\sum_{t=1}^T \mathbf{M}^{(t)} \right) + \eta T + \frac{\ln n}{\eta}$$

B. Solving an SDP with the MMW algorithm

A general SDP with n^2 variables (arranged into an $n \times n$ matrix \mathbf{X}) and m constraints can be written in the following form:

Primal	Dual
$\max \mathbf{C} \bullet \mathbf{X}$	$\min \mathbf{b} \cdot \mathbf{y}$
$\forall j \in [m]: \mathbf{A}_j \bullet \mathbf{X} \leq b_j$	$\sum_{j=1}^m \mathbf{A}_j y_j \succeq \mathbf{C}$
$\mathbf{X} \succeq \mathbf{0}$	$\mathbf{y} \geq \mathbf{0}$,

where $\mathbf{y} = (y_1, y_2, \dots, y_m)$ is the vector of dual variables and $\mathbf{b} = (b_1, b_2, \dots, b_m)$. The matrix \mathbf{X} and vector \mathbf{y} are (respectively) primal and dual feasible if they satisfy the second row of the Primal and Dual problems. We assume that $\mathbf{A}_1 = \mathbf{I}$ and $b_1 = R$ yielding the constraint $\text{Tr}(x) \leq R$. With this choice, Slater's condition is satisfied for the dual program, thus strong duality holds and the optima of the two programs coincide.

The main idea of the algorithm is to use the MMW algorithm to find either a primal feasible solution of value greater than α or a dual feasible solution of value at most $\alpha + \varepsilon$ for any value α and error parameter $\varepsilon > 0$. Using this algorithm in conjunction with binary search, we can compute the optimum value to the required accuracy.

We will always maintain a candidate solution for the primal problem in the form of $\exp(\mathbf{A})$ for some matrix \mathbf{A} which

may not satisfy the other primal constraints. We'll improve this candidate by an ORACLE¹⁷ which essentially checks for violations of primal feasibility for a given candidate \mathbf{X} .

Definition 4.1: ORACLE is an algorithm that, given a primal candidate solution $\mathbf{X} \succeq 0$ either outputs a vector \mathbf{y} (if there is such a vector) such that

$$\begin{aligned} \mathbf{b} \cdot \mathbf{y} &\leq \alpha \\ \sum_{j=1}^m (\mathbf{A}_j \bullet \mathbf{X}) y_j &\geq \mathbf{C} \bullet \mathbf{X} \\ \mathbf{y} &\geq 0 \end{aligned}$$

our outputs "fail". In the latter case, we'll say that the `textsc(Oracle)` failed.

If \mathbf{X} is primal feasible, then

$$\mathbf{C} \bullet \mathbf{X} \leq \sum_{j=1}^m (\mathbf{A}_j \bullet \mathbf{X}) y_j \leq \sum_{j=1}^m b_j y_j \leq \alpha,$$

meaning that if ORACLE finds a vector \mathbf{y} , then \mathbf{X} is either primal infeasible or has value at most α . The ORACLE is useful because of the following lemma¹⁸:

Lemma 4.1: If the ORACLE outputs fail, meaning that no vector \mathbf{y} satisfies the conditions in the definition of ORACLE, then a suitably scaled version of \mathbf{X} is a primal feasible solution of value greater than α .

Of course the ORACLE is an algorithm in itself, which largely depends on the exact problem we want to approximate. This means that we have to define¹⁹ a parameter of the ORACLE which can measure it's effectiveness:

Definition 4.2: The width of ORACLE is the smallest $\rho \geq 0$ such that for every primal candidate \mathbf{X} , the vector \mathbf{y} returned by oracle satisfies $\|\mathbf{A}_j y_j - \mathbf{C}\| \leq \rho$.

This width is the scaling factor used to ensure that the loss matrices have a spectral norm of at most 1. It also shows the effectiveness of the ORACLE in helping the algorithm make progress: the lower the width, the faster the progress.

If the ORACLE ever fails, then the primal candidate is feasible. The following theorem²¹ shows that in the other case, a dual feasible solution can be constructed with value at most $\alpha + \varepsilon$:

Theorem 4.2: Run the Primal-Dual SDP solver with MMW with parameter setting $\eta = \frac{\varepsilon}{2\rho R}$ and $T = \lceil \frac{4\rho^2 R^2 \ln n}{\varepsilon^2} \rceil$ and assume that ORACLE never fails in any of the T iterations. Then the dual solution output, $\bar{\mathbf{y}}$ is feasible with objective value at most $\alpha + \varepsilon$.

The parameter values show that the number of iterations taken depends both on the width of ORACLE and the value of error we can allow. The wider ORACLE is and the smaller the error we allow, the slower the algorithm will be. A framework can be given for an SPD primal oracle which is similar to the framework of the SDP solver²².

If all the matrices in the SDP are diagonal, then the SDP

Algorithm 6 Primal-Dual SDP solver with MMW²⁰

- 1: Fix trace bound R , width bound ρ , parameters $\eta \leq 1$ and T for the MMW algorithm
 - 2: **for** $t = 1, 2, \dots, T$ **do**
 - 3: Obtain the density matrix $\mathbf{P}^{(t)}$ from step t of the MMW algorithm
 - 4: $\mathbf{X}^{(t)} = R\mathbf{P}^{(t)}$
 - 5: Run ORACLE with primal candidate $\mathbf{X}^{(t)}$
 - 6: **if** ORACLE fails **then**
 - 7: Abort.
 - 8: **end if**
 - 9: Let $\mathbf{y}^{(t)}$ be the vector generated by ORACLE
 - 10: Use $\mathbf{M}^{(t)} = \frac{1}{\rho} (\sum_{j=1}^m \mathbf{A}_j y_j^{(t)} - \mathbf{C})$ as the loss matrix in the next step of the MMW algorithm
 - 11: **end for**
 - 12: If ORACLE never fails in all T rounds, then output the dual solution $\bar{\mathbf{y}} = \frac{1}{T} \sum_{t=1}^T \mathbf{y}^{(t)} + \frac{\varepsilon}{R} \mathbf{e}_1$.
-

reduces to an LP, and our algorithm reduces to the standard MW algorithm used for LPs.

V. CONCLUSION, FURTHER PLANS

In conclusion, the Multiplicate Weights method can be used to solve Linear programs. Furthermore, there is a generalization of the Multiplicate Weights method of solving Linear programs which can solve Semidefinite programs efficiently, however it's running time depends on the width of the ORACLE it's using, and the margin of error we allow.

The way in which this Primal-Dual SDP solver is designed means that it could be quite useful in solving Semidefinite programs which might occur in quantum computation.

My next goal is to discover the basics of quantum computation and then learn about the quantum generalization of Markov chains. Once that is done, I shall explore certain quantum computing problems which could be optimized combining the quantum generalization of Markov Chains and the Primal-Dual SPD solver algorithm.

REFERENCES

- [1] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications*, subsection 1.1, p. 124 <https://theoryofcomputing.org/articles/v008a006/>
- [2] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications*, p. 125 <https://theoryofcomputing.org/articles/v008a006/>
- [3] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications*, Theorem 1.1, p. 124 <https://theoryofcomputing.org/articles/v008a006/>
- [4] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications*, Theorem 2.1, p. 126 <https://theoryofcomputing.org/articles/v008a006/>
- [5] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications*, Theorem 2.1, p. 126 <https://theoryofcomputing.org/articles/v008a006/>
- [6] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications*, Corollary 2.2, p. 128 <https://theoryofcomputing.org/articles/v008a006/>

- [7] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications, Theorem 2.3*, p. 129 <https://theoryofcomputing.org/articles/v008a006/>
- [8] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications, Theorem 2.4*, p. 130 <https://theoryofcomputing.org/articles/v008a006/>
- [9] S. ARORA - E. HAZAN - S. KALE *The Multiplicative Weights Update Method: A Meta-Algorithm and Applications, Theorem 2.4*, p. 130 <https://theoryofcomputing.org/articles/v008a006/>
- [10] J. VAN APELDOORN - A. GILYÉN *Quantum algorithms for zero-sum games, Main Algorithm*, p. 3 <https://arxiv.org/pdf/1904.03180>
- [11] J. VAN APELDOORN - A. GILYÉN *Quantum algorithms for zero-sum games, Lemmas 1 and 3, Claim 2*, p. 3 – 6 <https://arxiv.org/pdf/1904.03180>
- [12] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs, subsection 2.3*, p. 6 <https://dl.acm.org/doi/pdf/10.1145/2837020>
- [13] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs*, p. 7 <https://dl.acm.org/doi/pdf/10.1145/2837020>
- [14] J. VAN APELDOORN - A. GILYÉN - S. GRIBLING - R. DE WOLF *Quantum SDP-solvers: Better upper and lower bounds*, p. 12 <https://arxiv.org/pdf/1705.01843>
- [15] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs, Theorem 3.1.*, p. 8 <https://dl.acm.org/doi/pdf/10.1145/2837020>
- [16] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs, Theorem 3.1.*, p. 8 <https://dl.acm.org/doi/pdf/10.1145/2837020>
- [17] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs, Definition 4.1.*, p. 10 <https://dl.acm.org/doi/pdf/10.1145/2837020>
- [18] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs, Lemma 4.2.*, p. 10 <https://dl.acm.org/doi/pdf/10.1145/2837020>
- [19] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs, Definition 4.3.*, p. 11 <https://dl.acm.org/doi/pdf/10.1145/2837020>
- [20] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs, Algorithm 2*, p. 11 <https://dl.acm.org/doi/pdf/10.1145/2837020>
- [21] S. ARORA - S. KALE *A Combinatorial, Primal-Dual Approach to Semidefinite Programs, Definition 4.4.*, p. 12 <https://dl.acm.org/doi/pdf/10.1145/2837020>