Project Report

Melissa Gaddy, Zheming Gao

December 5, 2017

1 Matrix Multiplicative Weights Algorithm and its application

1.1 Matrix MW Algorithm

The motivation of Matrix MW Algorithm is similar to MW algorithm introduced in the preceding section.

Imagine a 2-player zero-sum game. The first player chooses a unit vector $\mathbf{v}$, and the second player chooses a matrix M such that $\mathbf{0} \preceq M \preceq I$. The second player has to pay $\mathbf{v}^T M \mathbf{v}$ to the first.We allow the first player to choose the vector v from distribution $\mathcal{D}$ over the unit ball. In this case, we are interested in studying the value

$$\mathbb{E}_{\mathcal{D}}\left[(\mathbf{v}^\top M \mathbf{v})\right] = M \bullet \mathbb{E}_{\mathcal{D}}\left[\mathbf{v}\mathbf{v}^\top\right]$$

Now the matrix $P := \mathbb{E}_{\mathcal{D}}\left[\mathbf{v}\mathbf{v}^\top\right]$ is a density matrix, which has properties of positive semidefinite and trace 1.

As in MW algorithm, we consider an online 2-player zero-sum game. At each round the first player has to react to the opponent who picks matrix M. The matrix M is called an *observed event*. An online algorithm for first player to observe the event $M^{(t)}$ and choose a density matrix $P^{(t)}$ at each round $t = 1, \ldots, T$. After T rounds the best fixed vector $\mathbf{v}$ is the one that maximize the total payoff $\sum_{t=1}^{T} \mathbf{v}^\top M \mathbf{v}$.

For a symmetric matrix $A \in \mathbb{R}^n$, we denote its eigenvalues in an order as $\lambda_1(A) \geqslant \cdots \geqslant \lambda_n(A)$. Then it is obvious to see that the best maximizer $\mathbf{v}$ is the unit eigenvector of the largest eigenvalue of $\sum_{t=1}^{T} M^{(t)}$, i.e., $\lambda_1(\sum_{t=1}^{T} M^{(t)})$. The goal of this online algorithm is to find a solution vector $\mathbf{v}$ such that the value of $\mathbf{v}^\top M \mathbf{v}$ is not much less than the smallest expected value. We won't know which fixed vector $\mathbf{v}$ will be the best until we finish the last round. Hence, this algorithm has the advantage to make decisions while the game is playing.

Analogously, the Matrix MW algorithm is the following.

Algorithm 1 Matrix Multiplicative Weights Algorithm

Fix an $\epsilon < \frac{1}{2}$, and let $\eta = -\ln(1-\epsilon)$.

for $t = 1, 2, \ldots, T$ **do**

1. Compute weight matrix $W^{(t)} = \exp(\eta \sum_{\tau=1}^{t-1} M^{(\tau)})$.
2. Compute density matrix $P^{(t)} = \frac{W^{(t)}}{Tr(W^{(t)})}$.
3. Observe the event $M^{(t)}$.

1.2 Solving SDP with Matrix MW Algorithm

We've introduced Multiplicative Weights(MW) algorithm and apply it on solving Linear programming problems of some certain form. We refer to the setting as *basic* or *scalar* case. Now we are considering some problem which has enough structure that we can obtain an analogous algorithm for them.

Moving from vectors to matrices, and from probability vector to density matrix, now we refer to the current setting as *matrix* case.

$$\begin{aligned} a_j^T \mathbf{x} &\geqslant 0 & A_j \bullet X &\geqslant 0 \\ \mathbf{1}^T \mathbf{x} &= 1 & Tr(X) &= 1 \\ \mathbf{x} &\geqslant 0. & X &\succeq 0. \end{aligned}$$

Notice that the feasibility problem on right hand side is a SDP problem. We are interested in using Matrix MW Algorithm to solve SDP problems of this certain form. As before, to fit the algorithm on this problem, we need to slightly modify Matrix MW Algorithm. Let parameter $\rho = \max_j \|A_j\|$. At each round t, set the observe event (gain matrix) $M^{(t)} = A_j/\rho$.

Algorithm 2 Matrix Multiplicative Weights Algorithm - SDP

Fix an $\epsilon < \frac{1}{2}$, and let $\eta = -\ln(1-\epsilon)$. $\rho = \max_j \|A_j\|$.

Initialize $W^{(0)} = I$, $X^{(0)} = \frac{1}{n} I$.

while $A_j \bullet X^{(t)} < 0$ **do**

1. Observe gain matrix $M^{(t)} = \frac{A_j}{\rho}$.
2. Compute weight matrix $W^{(t)} = \exp(\eta \sum_{\tau=1}^{t} M^{(\tau)})$.
3. Compute density matrix $X^{(t)} = \frac{W^{(t)}}{Tr(W^{(t)})}$.

The Matrix Multiplicative Weights Algorithm - SDP gives an lower bound of the gain in total.

Theorem 1. *The Matrix Multiplicative Weights algorithm generates $X^{(t)}$, $t = 1, \ldots, T$ such that:*

$$\sum_{t=1}^{T} M^{(t)} \bullet X^{(t)} \geqslant (1+\epsilon)\lambda_1 \left(\sum_{t=1}^{T} M^{(t)} \right) - \frac{\ln n}{\epsilon}$$

(Add reference, A Combinatorial, Primal-Dual approach)

****

1.3 Max Cut problem

The max cut problem can be reformulate as the following form

$$\begin{array}{lll} \max & \frac{1}{4}L \bullet X & \\ \text{s.t.} & (e_j e_j^T) \bullet X & = 1, \quad \forall j = 1, \ldots, n \\ & X & \succeq 0. \end{array} \tag{1}$$

where $X \in \mathbb{S}_+^n$.

If we denote the optimal value as b and scale X by $1/n$, then we would like to check the feasibility of the constraints:

$$\begin{array}{ll} \frac{n}{4b}L \bullet X & \geqslant 1 \\ n(e_j e_j^T) \bullet X & = 1, \quad \forall j = 1, \ldots, n \\ X & \succeq 0. \end{array}$$

Finally, if we take $A_j = n(e_j e_j^T) - I, (j = 1, \ldots, n)$, and $A_{n+1} = \frac{n}{4b}L - I$, then the feasibility problem can be reformed as

$$\begin{array}{lll} A_j \bullet X & \geqslant 0 & \forall j = 1, \ldots, n+1 \\ \mathrm{Tr}(X) & = 1 & \\ X & \succeq 0. & \end{array} \tag{2}$$

on which the Matrix-MW algorithm can be applied. To apply Matrix-MW, we would like to solve the large-margin problem with an error parameter $\delta > 0$. Denote $K = \{X \succeq 0 | \mathrm{Tr}(X) = 1\}$, we either solve the problem to an additive error of δ, i.e., find $X \in K$ such that $A_j \bullet X \geqslant -\delta, \forall j = 1, \ldots, n+1$, or prove the system (2) is infeasible.

Initialize $X^{(1)} = 1/nI$, and set parameters ϵ, $\eta = -\ln(1-\epsilon)$, we apply Matrix-MW on three different examples. For each example, we list the result and errors for different parameters.

1.3.1 Results

ϵ	0.1	0.01	0.001	0.0001
δ	0.1	0.01	0.001	0.0001
# of rounds	20	328	3465	34827
Rank(X)	4	4	4	4
$\|X - X^*\|_2$	2.7700	2.7557	2.7556	2.7545
$\|X - X^*\|_2 / \|X^*\|_2$	0.6925	0.6889	0.6887	0.6886
$\|b^* - b\|$	0.4595	0.0482	0.0039	0.0002

Table 1: toy example

1.3.2 Comments

1. "# or rounds" is the number of iterations for Matrix-MW algorithm. It stops when $A_j \bullet X \geqslant -\delta$ is satisfied for any $j = 1, \ldots, n+1$.

ϵ	0.1	0.01	0.001	0.0001
δ	0.1	0.01	0.001	0.0001
# of rounds	19	633	7230	73228
Rank(X)	10	10	10	10
$\|X - X^*\|_2$	8.9298	8.9171	8.9169	8.9168
$\|X - X^*\|_2/\|X^*\|_2$	0.8930	0.8917	0.8917	0.8917
$\|b^* - b\|$	1.9107	0.1908	0.0204	0.0011

Table 2: 10 nodes example

ϵ	0.1	0.01	0.001	0.0001
δ	0.1	0.01	0.001	0.0001
# of rounds	2777	49971		
$\|b^* - b\|$	3.2430	0.7318		

Table 3: 100 nodes example

2. Solution X and the value of $b = \frac{1}{4}L \bullet X$ are obtained from iteration. X^* and b^* are optimal solution and optimal value to problem (1). We list the error of solution $\|X - X^*\|_2$ and the error of object value $|b - b^*|$.

3. Notice the solution absolute error in 2-norm is very large, so we also calculate the relative error in 2-norm for the toy example and the 10-node example.

4. The 100-node example has blank units in table since the algorithm converges very slow when ϵ and δ are smaller than 10^{-2}. (# of iteration will exceed 10^5).

5. Rank of the iteration solution is not 1. So the Rank = 1 constraint of original MAX CUT problem cannot be eliminated.

6. Notice that for each result, the ϵ and δ are in the same order of magnitude. The reason is, if we set ϵ and δ in different orders of magnitude, e.g., $\epsilon = 0.01$ and $\delta = 0.001$, or $\epsilon = 0.001$ and $\delta = 0.01$, the algorithm will not converge or converge very slow. From this we conclude that Matrix-MW is very parameter sensitive.