ORIE 6334 Bridging Continuous and Discrete Optimization	Sept. 16, 2019
Lecture 4	
Lecturer: David P. Williamson	*Scribe: Yicheng Bai*

1 Recap

1.1 The Multiplicative Weights Algorithm

Assume there are N possible different decisions that could be made at each time. Let $v_t(i) \in [0,1]$ denote the value of making decision i at time t. We maintain a weight $w_t(i)$ as a weight associated with decision i at time t. Also let

$$W_t = \sum_{i=1}^{N} w_t(i), \qquad p_t(i) = \frac{w_t(i)}{W_t}.$$

We have the multiplicative weights algorithm as follows:

Algorithm 1: Multiplicative Weights

$w_1(i) \leftarrow 1,\ \forall i = 1, \ldots, N$
for $t \leftarrow 1$ **to** T **do**
  Pick decision i with probability $p_t(i)$ and get value $v_t(i)$
  $w_{t+1}(i) \leftarrow (1 + \epsilon v_t(i)) w_t(i),\ \forall i = 1, \ldots, N$
end

We know from last lecture that the expected revenue gained by Algorithm 1 is close enough to the value one can gain from the best fixed decision in hindsight. Specifically, we have the following theorem.

Theorem 1 *Assume $\epsilon \le 1/2$, then for all j,*

$$\sum_{t=1}^{T} \sum_{i=1}^{N} p_t(i) v_t(i) \ge (1 - \epsilon) \sum_{t=1}^{T} v_t(j) - \frac{1}{\epsilon} \ln N.$$

1.2 Application: Finding ϵ-Feasible Solution

We showed how to apply the multiplicative weights algorithm to finding ϵ-feasible solutions to the following system:

$$Ax \le e, \quad x \in Q. \tag{1}$$

where $A \in \mathbb{R}^{m \times n}$, $e \in \mathbb{R}^m$ is the vector of all ones, and $Q \subseteq \mathbb{R}^n$ is a convex set. Assume $Ax \ge 0$ for each $x \in Q$.

We also assume that we have an oracle such that given vector $p \in \mathbb{R}^m_{\ge 0}$, finds $x \in Q$ such that $p^T Ax \le p^T e$, if such an x exists.

Define width of oracle to be:

$$\rho := \max_{i=1,\ldots,m} \max_{\substack{x \in Q \\ \text{returned} \\ \text{by oracle}}} (Ax)(i).$$

By building on the multiplicative weights algorithm, there is an algorithm for this problem (Algorithm 2 in the last lecture) and we have the following theorem:

Theorem 2 *The algorithm (Algorithm 2 in the last lecture) finds $\overline{x} \in Q$ s.t. $A\overline{x} \leq (1+4\epsilon)e$ in time*

$$O\left(\frac{m\rho}{\epsilon^2} \ln m\right) + O\left(\frac{\rho}{\epsilon^2} \ln m\right)(\textit{oracle calls} + \textit{matrix multiplication}).$$

2 Application: Max Flow in Unit Capability Graphs

To see an application of the result of the last section, we consider the maximum flow problem in directed graps. Let $G = (V, E)$ be a directed graph, let $s \in V$ be the source and $t \in V$ be the sink. The capacities are $u(i,j) = 1$, $\forall (i,j) \in E$. The goal is to find the maximum flow from source s to sink t in this directed graph G.

A maximum flow is an optimization problem, and the results of the previous section only check feasibility of a system. How can we reduce this optimization problem to a feasibility problem? The idea is to check if there is a feasible flow of value k. Specifically, we can use binary search to find max flow value, since the value is at most $m = |E|$, we only need $\lceil \log_2 m \rceil$ calls checking for a feasible flow.

Now we show how to reduce the problem of determining whether there exists a feasible flow of value k to the framework given above. We let A be capacity constraints such that $x(i,j) \leq 1$ for all $(i,j) \in E$, hence A is the identity matrix. We let Q be flow conservation constraints, and the flow value. We have that:

$$Q = \left\{ x \geq 0 : \sum_{j:(i,j)\in E} x(i,j) - \sum_{j:(j,i)\in E} x(j,i) = 0, \forall i \neq s,t; \sum_{j:(s,j)\in E} x(s,j) - \sum_{j:(j,s)\in E} x(j,s) = k \right\}.$$

So there exists $x \in Q$ such that $Ax \leq e$ iff a feasible flow of value k exists.

We now need an oracle that checks if $\exists x \in Q$ s.t. $p^T Ax \leq p^T e$ for $p \geq 0$. Note that in this case $p^T Ax = \sum_{(i,j)\in E} p(i,j)x(i,j)$. Then we can directly find $x \in Q$ that minimizes $p^T Ax = \sum_{(i,j)\in E} p(i,j)x(i,j)$ by finding shortest $s-t$ path when using $p(i,j) \geq 0$ as lengths, and sending k units of flow on this path. Since all the lengths $p(i,j)$ are non-negative, we can use Dijkstra's algorithm in $\mathcal{O}(m + nlogn)$ time to find the shortest path.

Here, the oracle width is $\rho \leq k \leq m$.

So the running time would be (by Theorem 2):

$$\mathcal{O}\left(\frac{m^2}{\epsilon^2} \ln m + \frac{m}{\epsilon^2}(m + n \log n)\right) = \tilde{\mathcal{O}}\left(\frac{m^2}{\epsilon^2}\right).$$

This running time is not very good for this problem. There are classical flow algorithms finding exact solutions (rather than approximate ones) in $O(m^{3/2})$ time (and faster algorithms have been found recently). However, the point was to illustrate a use for the algorithm of the previous section.

3 Application: Max Multicommodity Flow

We now turn to another application of the multiplicative weight algorithm, now to the maximum multicommodity flow problem. Let $G = (V, E)$ be a directed graph. There are K source-sink pairs $s_1 - t_1$, $s_2 - t_2$, $\cdots$, $s_K - t_K$. $u(i,j) \geq 0$ represents the capacity of an edge. The goal is to find $s_i - t_i$ flow f_i for each i that maximizes $\sum_{k=1}^{K}$(value of flow f_i) subject to the constraints that $\sum_{k=1}^{K} f_k(i,j) \leq u(i,j)$, for all $(i,j) \in E$.

Note that there is nothing similar to integrality property or max-flow min-cut theorem for this problem.

Let $\mathcal{P}_k$ = set of all $s_k - t_k$ paths and $\mathcal{P} = \cup_{k=1}^{K} \mathcal{P}_k$.

We can formulate this problem in terms of linear programming, in which we have a variable $x(P)$ representing the amount of flow sent on path $P \in \mathcal{P}$.

$$\begin{aligned} \max \quad & \sum_{P \in \mathcal{P}} x(P) \\ \text{s.t.} \quad & \sum_{P \in \mathcal{P} : (i,j) \in P} x(P) \leq u(i,j) \\ & x(P) \geq 0 \end{aligned} \tag{P}$$

Its dual can be written as:

$$\begin{aligned} \min \quad & \sum_{(i,j) \in E} u(i,j) \ell(i,j) \\ \text{s.t.} \quad & \sum_{(i,j) \in P} \ell(i,j) \geq 1 \quad \text{for any } P \in \mathcal{P} \\ & \ell(i,j) \geq 0 \end{aligned} \tag{D}$$

Then we have the following algorithm[1] for this problem as follows:

Let P_t be path chosen in iteration t, $w_t(i,j)$ be weights in iteration t, u_t be u in iteration t, $w_t = \sum_{(i,j) \in E} w_t(i,j)$, and T be number of iterations. Let X be the value of flow we compute, which is $\sum_{t=1}^{T} u_t = \sum_{P \in \mathcal{P}} x(P)$. Let X^* = value of max flow.

First observe that this algorithm does not compute a feasible flow; the value of flows on edges can and will be larger than the capacity of the edges. We will explain later how to find a feasible solution by scaling all the flows down by the same value.

Next observe that Algorithm 2 looks like multiplicative weights algorithm when

$$v_t(i,j) = \begin{cases} \frac{u_t}{u(i,j)}, & \forall\ (i,j) \in P_t \\ 0, & otherwise \end{cases}$$

$$p_t(i,j) = \frac{w_t(i,j)}{W_t}$$

[1] This algorithm was first proposed by Garg and Könemann 1998 `http://pure.mpg.de/rest/items/item_1819555/component/file_2574820/content` and then restated by Arora, Hazan, and Kale 2012 `http://theoryofcomputing.org/articles/v008a006/v008a006.pdf`.

Algorithm 2: Find Max Multicommodity Flow

$x(P) \leftarrow 0,\ \forall P \in \mathcal{P}$

$f(i,j) \leftarrow 0,\ w(i,j) \leftarrow 1,\ \forall (i,j) \in E$

while $\frac{f(i,j)}{u(i,j)} < \frac{\ln m}{\epsilon^2},\ \forall (i,j) \in E$ **do**

Find $P \in \mathcal{P}$ that minimizes $\sum_{(i,j)\in P} \frac{w(i,j)}{u(i,j)}$ (*which can be done by computing $s_k - t_k$ shortest path for all k using lengths* $\frac{w(i,j)}{u(i,j)}$)

$u \leftarrow \min_{(i,j)\in P} u(i,j)$

$x(P) \leftarrow x(P) + u$

$f(i,j) \leftarrow f(i,j) + u,\ \forall (i,j) \in P$

$w(i,j) \leftarrow (1 + \epsilon \frac{u}{u(i,j)}) w(i,j),\ \forall (i,j) \in P$

end

.

Given that this is the case, we can simply apply Theorem 1, and obtain that for all edges $(h,k) \in E$:

$$\begin{aligned} \sum_{t=1}^{T} \sum_{(i,j)\in P_t} \frac{u_t}{u(i,j)} \frac{w_t(i,j)}{w_t} &\geq (1-\epsilon) \sum_{t=1}^{T} \frac{u_t}{u(h,k)} \mathbb{1}_{(h,k)\in P_t} - \frac{1}{\epsilon} \ln m \\ &= (1-\epsilon) \frac{f(h,k)}{u(h,k)} - \frac{1}{\epsilon} \ln m. \end{aligned} \tag{1}$$

Now consider the dual solution for iteration t:

$$\ell_t(i,j) = \frac{\frac{w_t(i,j)}{u(i,j)}}{\sum_{(a,b)\in P_t} \frac{w_t(a,b)}{u(a,b)}}.$$

The solution is feasible, since for any path $P \in \mathcal{P}$:

$$\sum_{(i,j)\in P} \ell_t(i,j) = \frac{\sum_{(i,j)\in P} \frac{w_t(i,j)}{u(i,j)}}{\sum_{(a,b)\in P_t} \frac{w_t(a,b)}{u(a,b)}} \geq 1.$$

where the inequality holds since P_t is the shortest path at iteration t. Since the dual objective function value is always an upper bound of the primal value, we have:

$$\begin{aligned} X^* &\leq \sum_{(i,j)\in E} u(i,j) \ell_t(i,j) \\ &= \frac{\sum_{(i,j)\in E} w_t(i,j)}{\sum_{(a,b)\in P_t} \frac{w_t(a,b)}{u(a,b)}} \\ &= \frac{W_t}{\sum_{(a,b)\in P_t} \frac{w_t(a,b)}{u(a,b)}}. \end{aligned} \tag{2}$$

Thus, the left hand side of inequality (1) can be written as:

$$\begin{aligned}\sum_{t=1}^{T}\sum_{(i,j)\in P_t}\frac{u_t}{u(i,j)}\frac{w_t(i,j)}{W_t} &= \sum_{t=1}^{T}\frac{u_t}{W_t}\sum_{(i,j)\in P_t}\frac{w_t(i,j)}{u(i,j)}\\ &\le \frac{1}{X^*}\sum_{t=1}^{T}u_t\\ &= \frac{X}{X^*}. \end{aligned} \tag{3}$$

where the inequality follows from (2).

Combining (1) and (3), we have that for all $(h,k)\in E$:

$$\frac{X}{X^*}\ge(1-\epsilon)\frac{f(h,k)}{u(h,k)}-\frac{1}{\epsilon}\ln m.$$

Let $C=\max\limits_{(h,k)\in E}\frac{f(h,k)}{u(h,k)}$, we have that $C\ge\frac{\ln m}{\epsilon^2}$ by the termination criterion of the while loop in algorithm 2.

Let $\tilde{x}(P)=\frac{x(P)}{C}$ for all $P\in\mathcal{P}$; we claim that then $\tilde{x}$ is a feasible solution to the primal problem. With the notion of C, we can further have:

$$\begin{aligned}\frac{X}{X^*}&\ge(1-\epsilon)C-\frac{1}{\epsilon}\ln m\\ &\ge(1-\epsilon)C-\epsilon C\\ &=(1-2\epsilon)C.\end{aligned}$$

So our feasible flow has value $\frac{X}{C}\ge(1-2\epsilon)X^*$.

For running time analysis of Algorithm 2:

1. Each edge can be the edge s.t. $u(i,j)=u$ at most $\frac{1}{\epsilon^2}\ln m$ times. Thus there are at most $\mathcal{O}(\frac{m\ln m}{\epsilon^2})$ iterations.

2. Within each iteration, we need to find K shortest paths.

3. Thus, the total running time is $\mathcal{O}(\frac{Km}{\epsilon^2}(m+n\log n))$. Fleishcher (2000) eliminated the dependency on K and improved the running time to be $\mathcal{O}(\frac{m}{\epsilon^2}(m+n\log n))$ `http://epubs.siam.org/doi/pdf/10.1137/S0895480199355754`.