Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Multiplicative Weights Intro

Multiplicative Weights

This is an online algorithm, in which you take into account the advice of n experts. Every day you get more information on how good every expert is until the last day T.

Let's first define some terminology:

- $x_i^{(t)}$ = proportion that you 'trust' expert i on day t
- $l_i^{(t)}$ = loss you would incur on day i if you invested everything into expert i
- total regret: $R_T = \sum_{t=1}^{T} \sum_{i=1}^{n} x_i^{(t)} l_i^{(t)} - \min_{i=1,\ldots,n} \sum_{t=1}^{T} l_i^{(t)}$

$\forall i \in [1, n]$ and $\forall t \in [1, T]$, the multiplicative update is as follows:

$$w_i^{(0)} = 1$$

$$w_i^{(t)} = w_i^{(t-1)}(1-\epsilon)^{l_i^{(t-1)}}$$

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

If $\epsilon \in (0, 1/2]$, and $l_i^{(t)} \in [0, 1]$, we get the following bound on total regret:

$$R_T \leq \epsilon T + \frac{\ln(n)}{\epsilon}$$

Let's play around with some of these questions. For this problem, we will be running the randomized multiplicative weights algorithm with two experts. Consider every subpart of this problem distinct from the others.

(a) Let's say we believe the best expert will have cost 20, we run the algorithm for 100 days, and epsilon is $\frac{1}{2}$. What is the maximum value that the total loss incurred by the algorithm can be?

(b) What value of ϵ should we choose to minimize the total regret, given that we run the algorithm for 25 days?

(c) We run the randomized multiplicative weights algorithm with two experts. In all of the first 140 days, Expert 1 has cost 0 and Expert 2 has cost 1. If we chose $\epsilon = 0.01$, on the 141st day with what probability will we play Expert 1? (Hint: You can assume that $0.99^{70} = \frac{1}{2}$)

Solution:

(a) total regret = loss of algorithm - offline optimum $\leq \epsilon T + \frac{\ln(n)}{\epsilon}$.

$$\text{loss of algorithm} - 20 \leq \frac{1}{2}(100) + \frac{\ln(2)}{\frac{1}{2}}$$

$$\text{loss of algorithm} \leq 50 + 2\ln(2) + 20$$

The maximum loss is roughly 71.39.

(b)

$$R_T \leq \epsilon T + \frac{\ln(n)}{\epsilon}$$

$$R_T \leq \epsilon 25 + \frac{\ln(2)}{\epsilon}$$

Take the derivative with respect to epsilon and set the derivative to 0 to get:

$$25 - \frac{\ln(2)}{\epsilon^2} = 0$$

$$25 = \frac{\ln(2)}{\epsilon^2}$$

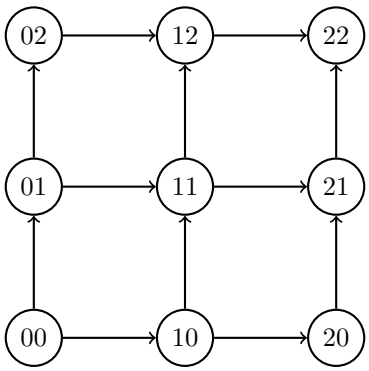
$$\epsilon = \sqrt{\frac{\ln(2)}{25}}$$

ϵ should be roughly 0.17.

(c) The weight assigned to expert 1 is $.99^{0 \cdot 140} = 1$, while the weight assigned to expert 2 is $.99^{1 \cdot 140} \approx 1/4$. So, the probability we play expert 1 is $\frac{1}{1+1/4} = 4/5$.

2 Multiplicative Weights

Consider the following simplified map of Berkeley. Due to traffic, the time it takes to traverse a given path can change each day. Specifically, the length of each edge in the network is a number between $[0, 1]$ that changes each day. The travel time for a path on a given day is the sum of the edges along the path.



For T days, both Max and Vinay drive from node 00 to node 22.

To cope with the unpredictability of traffic, Vinay builds a time machine and travels forward in time to determine the traffic on each edge on every day. Using this information, Vinay picks the path that has the smallest total travel time over T days, and uses the same path each day.

Max wants to use the multiplicative weights update algorithm to pick a path each day. In particular, Max wants to ensure that the difference between his expected total travel time over T days and Vinay's total travel time is at most $T/10000$. Assume that Max finds out the lengths of all the edges in the network, even those he did not drive on, at the end of each day.

(a) How many experts should Max use in the multiplicative weights algorithm?

(b) What are the experts?

(c) Given the weights maintained by the algorithm, how does Max pick a route on any given day?

(d) The regret bound for multiplicative weights is as follows:

Theorem. Assuming that all losses for the n experts are in the range $[0, 4]$, the worst possible regret of the multiplicative weights algorithm run for T steps is

$$R_T \leq 8\sqrt{T \ln n}$$

Use the regret bound to show that expected total travel time of Max is not more than $T/10000$ worse than that of Vinay for large enough T.

Solution:

(a) 6 experts

(b) There is one expert for every path from 00 to 22. One can see that there are 6 different paths from 00 to 22.

(c) The algorithm maintains one weight for each path. Max picks a path with probability proportional to its weight.

(d) Let $P_1, \ldots, P_6$ be the 6 possible paths between 00 and 22. Let $\ell_i^{(t)}$ denote the length of path i on day t. Let $w_i^{(t)}$ denote the weight of path i on day t.
Since Max picks a path proportional to its weight, the expected total time on day t is

$$\sum_{i=1}^{6} w_i^{(t)} \cdot \ell_i^{(t)},$$

and the expected total time over T days is,

$$\sum_{t=1}^{T} \sum_{i=1}^{6} w_i^{(t)} \cdot \ell_i^{(t)}.$$

The regret bound to multiplicative weights asserts that for every path P_i,

$$\sum_{t=1}^{T} \sum_{i=1}^{6} w_i^{(t)} \cdot \ell_i^{(t)} - \sum_{t=1}^{T} \ell_i^{(t)} \leq 8\sqrt{T \ln 6}\ .$$

Here $n = 6$, since there are 6 different paths. Since Vinay picks one of the paths P_i to use over all days, the above regret bound implies that total expected time of Max is at most $8\sqrt{T \ln 6}$ worse than that of Vinay. For sufficiently large T, $8\sqrt{T \ln 6} < T/10000$, in particular $T > (8 \cdot 10000)^2 \cdot \ln 6$ suffices.

Reduction: Suppose we have an algorithm to solve problem A, how to use it to solve problem B?

This has been and will continue to be a recurring theme of the class. Examples so far include

- Use SCC to solve 2SAT.

- Use LP to solve max flow.
- Use max flow to solve mincut.
- Use max flow to solve maximum bipartite matching.

In each case, we would transform the instance of problem B we want to solve into an instance of problem A that we can solve. Importantly, the transformation is efficient, say, in polynomial time.

Conceptually, a efficient reduction means that problem B is no harder than A. On the other hand, if we somehow know that B cannot be solved efficiently, we cannot hope that A can be solved efficiently.

To show that the reduction works, you need to prove (1) if there is a solution for an instance of problem A, there must be a solution to the transformed instance of problem B and (2) if there is a solution to the transformed instance of B, there must be a solution in the corresponding instance of problem A.

3 Vertex Cover to Set Cover

In the minimum vertex cover problem, we are given an undirected unweighted graph $G = (V, E)$ and asked to find the smallest set $U \subseteq V$ that "covers" the set of edges E. In other words, we want to find the smallest set U such that for each $(u, v) \in E$, either u or v is in U.

Now recall the definition of the minimum set cover problem: Given a set U of elements and a collection $S_1, \ldots, S_m$ of subsets of U, the problem asks for the smallest collection of these sets whose union equals U.

Give an efficient reduction from the minimum vertex cover problem to the minimum set cover problem.

Solution: Let $G = (V, E)$ be an instance of the minimum vertex cover problem. Create an instance of the Minimum Set Cover Problem where $U = E$ and for each $u \in V$, the set S_u contains all edges adjacent to u. Let $C = \{S_{u_1}, S_{u_2}, \ldots, S_{u_k}\}$ be a set cover. Then our corresponding vertex cover will be $u_1, u_2, \ldots, u_k$. To see this is a vertex cover, take any $(u, v) \in E$. Since $(u, v) \in U$, there is some set S_{u_i} containing (u, v), so u_i equals u or v and (u, v) is covered in the vertex cover.

Now take any vertex cover $u_1, \ldots, u_k$. To see that $S_{u_1}, \ldots, S_{u_k}$ is a set cover, take any $(u, v) \in E$. By the definition of vertex cover, there is an i such that either $u = u_i$ or $v = u_i$. So $(u, v) \in S_{u_i}$, so $S_{u_1}, \ldots, S_{u_k}$ is a set cover.

Since every vertex cover has a corresponding set cover (and vice-versa) and minimizing set cover minimizes the corresponding vertex cover, the reduction holds.

4 Maximum Spanning Tree

In this class, we have been talking about minimum spanning tree. What about maximum spanning tree? Can you use the minimum spanning tree algorithms we learned, Prim's and Kruskal's, as blackbox to find maximum spanning tree? Assume the graph is undirected and with positive edge weights.

Solution: Negate the weights for each edge and apply the minimum spanning tree algorithm, say, Kruskal's.

The maximum spanning tree on the original graph is the minimum spanning tree of the negated graph. One can observe that the correctness of Kurskal's and Prim's is not affected by the negative weight edges. This is because the cut property still holds even if we are talking about negative weights.