Zero Sum Games: In this game, there are two players: a maximizer and a minimizer. We generally write the payoff matrix M in perspective of the maximizer, so every row corresponds to an action that the maximizer can take, every column corresponds to an action that the minimizer can take, and a positive entry corresponds to the maximizer winning. M is a n by m matrix, where n is the number of choices the maximizer has, and m is the number of choices the minimizer has.

A linear program that represents fixing the maximizer's choices to a probabilistic distribution where the maximizer has n choices, and the probability that the maximizer chooses choice i is p_i is the following:

$$\begin{gathered}
\max(z) \\
M_{1,1}(p_1) + \cdots + M_{n,1}(p_n) \geq z \\
M_{1,2}(p_1) + \cdots + M_{n,2}(p_n) \geq z \\
\vdots \\
M_{1,m}(p_1) + \cdots + M_{n,m}(p_n) \geq z \\
p_1 + p_2 + \cdots + p_n = 1 \\
p_1, p_2, \cdots, p_n \geq 0
\end{gathered}$$

The dual represents fixing the minimizers choices to a probabilistic distribution.

By strong duality, the optimal value of the game is the same if you fix the minimizer's distribution first or the maximizer's distribution first.

payoff matrix M:

(entries in perspective of the maximizer)

		minimizer		
		rock	paper	scissors
maximizer	rock	0	-1	1
	paper	1	0	-1
	scissors	-1	1	0

Writing the primal LP:

- maximize a payoff z
- find probability of maximizer playing each option: x_1, x_2, x_3

max (worst case minimizer move)

$$= \max \min \{ \overset{\text{rock}}{x_2 - x_3}, \overset{\text{paper}}{-x_1 + x_3}, \overset{\text{scissors}}{x_1 - x_2} \}$$

s.t. $x_1 + x_2 + x_3 = 1$

$x_1, x_2, x_3 \geq 0$

$\max z$

col chooses rock: $x_2 - x_3 \geq z$

col chooses paper: $-x_1 + x_3 \geq z$

col chooses scissors: $x_1 - x_2 \geq z$

$x_1 + x_2 + x_3 = 1$ probabilities sum to 1

$x_1, x_2, x_3 \geq 0$ and are nonnegative

Writing the dual LP:

- minimize payoff z
- find probability of minimizer playing each option: y_1, y_2, y_3

max (worst case minimizer move) = min (worst case maximizer move)

$$\max \min \{ x_2 - x_3, -x_1 + x_3, x_1 - x_2 \} = \min \max \{ -y_2 + y_3, y_1 - y_3, -y_1 + y_2 \}$$

$\min z$

$-y_2 + y_3 \leq z$

$y_1 - y_3 \leq z$

$-y_1 + y_2 \leq z$

$y_1 + y_2 + y_3 = 1$

$y_1, y_2, y_3 \geq 0$

1 Zero-Sum Games Short Answer

(a) Suppose a zero-sum game has the following property: The payoff matrix M satisfies $M = -M^\top$. What is the expected payoff of the row player?

↑ column player's payoff matrix

expected payoff for both players need to sum to 0

payoff for both players : 0

(b) True or False: If every entry in the payoff matrix is either 1 or -1 and the maximum number of 1s in any row is k, then for any row with less than k 1s, the row player's optimal strategy chooses this row with probability 0. Justify your answer.

False

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix}$$ ← pure strategy always causes row player to lose

column player sees no difference in cols

(c) True or False: Let M_i denote the ith row of the payoff matrix. If $M_1 = \frac{M_2+M_3}{2}$, then there is an optimal strategy for the row player that chooses row 1 with probability 0.

↖ linear combination, redundant

True

Assume optimal is p_1, p_2, p_3

can instead choose row 1 with probability 0

row 2 " " $p_2 + \frac{1}{2}p_1$

row 3 " " $p_3 + \frac{1}{2}p_1$

$$p_1M_1 + p_2M_2 + p_3M_3 = p_1\left(\frac{M_2+M_3}{2}\right) + p_2M_2 + p_3M_3$$

$$= \left(\frac{p_1}{2} + p_2\right)M_2 + \left(\frac{p_1}{2} + p_3\right)M_3$$

2 Permutation Games

A permutation game is a special form of zero-sum game. In a permutation game, the payoff matrix is n-by-n, and has the following property: Every row and column contains exactly the entries $p_1, p_2, \ldots p_n$ in some order. For example, the payoff matrix might look like:

$$P = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_2 & p_3 & p_1 \\ p_3 & p_1 & p_2 \end{bmatrix}$$

Given an arbitrary permutation game, describe the row and column players' optimal strategies, justify why these are the optimal strategies, and state the row player's expected payoff (that is, the expected value of the entry chosen by the row and column player).

Choose each row/col uniformly at random.

expected payoff : $\frac{1}{n}\sum_{k} p_k$

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 t 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:

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

$x_i^{(t)}$ = proportion of trust of expert i on day t

$\ell_i^{(t)}$ = loss of expert i on day t

$w_i^{(t)}$ = weight of expert i on day t (normalize weight to find $x_i^{(t)}$)

Multiplicative Weight Update Alg:

Start by weighting all experts equally $w_i^{(0)} = 1$

On day t, update weights by penalizing based on loss

$$w_i^{(t)} = w_i^{(t-1)}(1-\varepsilon)^{\ell_i^{(t-1)}} \qquad \varepsilon \in (0, \tfrac{1}{2}]$$

- ↳ experts with 0 loss don't decrease in weight
- ↳ experts with higher loss decreases more in weight
- ↳ larger ε means higher penalties

Normalize weights to find proportion of trust for each expert

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

On day t, pick expert i with probability $x_i^{(t)}$.

Regret:

R = (expected loss from algorithm) − (loss if we listened to the ONE overall best expert all throughout)

not necessarily 0, best expert can still be wrong some days

$$R = \sum_{t=1}^{T} \underbrace{\sum_{i=1}^{n} x_i^{(t)} \ell_i^{(t)}}_{\text{expected loss on day } t} - \min_i \underbrace{\sum_{t=1}^{T} \ell_i^{(t)}}_{\text{total loss of expert } i}$$

sum loss across all T days

If $\varepsilon \in [0, \frac{1}{2}]$ and $\ell_i^{(t)} \in [0,1]$, bound on total regret:

$$R \leq \varepsilon T + \frac{\ln(n)}{\varepsilon}$$

* proof in class notes

attendance: tinyurl.com/disc9170

3 Multiplicative Weights Intro

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?

$$R_T = \text{loss of alg} - \text{offline optimum} \leq \varepsilon T + \frac{\ln(n)}{\varepsilon}$$

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

$$\text{loss of alg} \leq 70 + 2\ln(2)$$

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

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

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

Take derivative:

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

$$25\varepsilon^2 = \ln(2)$$

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

(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 $0.99^{70} = \frac{1}{2}$)

$$w_1^{(0)} = w_2^{(0)} = 1$$

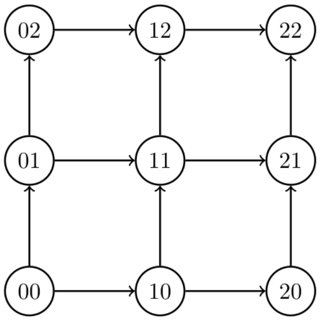
$$w_1^{(141)} = w_1^{(0)}\left[(1-\varepsilon)^0\right]^{140} = 1$$

$$w_2^{(141)} = w_1^{(0)}\left[(1-\varepsilon)^1\right]^{140} = 0.99^{140} = (0.99^{70})^2 = \left(\frac{1}{2}\right)^2 = \frac{1}{4}$$

$$\text{probability we play expert } 1 = x_1^{(141)} = \frac{1}{1 + 1/4} = 4/5$$

4 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 Tynan and Selina drive from node 00 to node 22.

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

offline optimum

Tynan wants to use the multiplicative weights update algorithm to pick a path each day. In particular, Tynan wants to ensure that the difference between his expected total travel time over T days and Selina's total travel time is at most $T/10000$. Assume that Tynan 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 Tynan use in the multiplicative weights algorithm?

6

$$\frac{4!}{2!2!} = 6$$

(b) What are the experts?

One expert for each path 00 to 22

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

pick paths with probability proportional to its weight (defined by MWU algorithm)

(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 Tynan is not more than $T/10000$ worse than that of Selina for large enough T.

$$R_T = \sum_{t=1}^{T} \sum_{i=1}^{6} x_i^{(t)} \ell_i^{(t)} - \min_i \sum_{t=1}^{T} \ell_i^{(t)} \leq 8\sqrt{T \ln 6}$$

$$8\sqrt{T \ln 6} \leq \frac{T}{10000}$$

$$80000^2 T \ln 6 \leq T^2$$

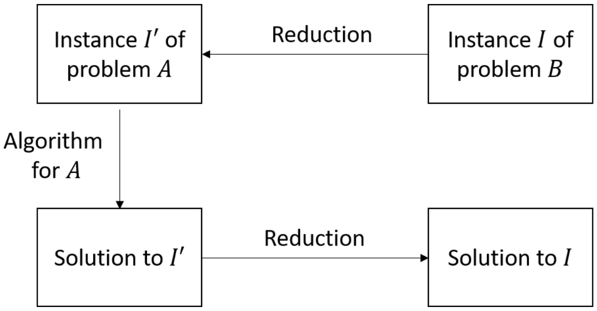
$$T \geq 80000^2 \ln 6$$

Reduction: Suppose we have an algorithm to solve problem A, how can we 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 LP to solve max flow.
- Use max flow to solve min s-t cut.
- Use minimum spanning tree to solve maximum spanning tree.
- Use Huffman tree to solve twenty questions.

In each case, we would transform the instance I of problem B we want to solve into an instance I' of problem A that we can solve, and also describe how to take a solution for I' and transform it into a solution for I:



Importantly, the transformation should be efficient, i.e. takes polynomial time. If we can do this, we say that we have reduced problem B to problem A.

Conceptually, a efficient reduction means that if we can solve problem A efficiently, we can also solve problem B efficiently. On the other hand, if we think that B cannot be solved efficiently, we also think that A cannot be solved efficiently. Put simply, we think that A is "at least as hard" as B to solve.

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

5 Some Sums

Given an array $A = [a_1, a_2, \ldots, a_n]$ of nonnegative integers, consider the following problems:

1 **Partition**: Determine whether there is a subset $P \subseteq [n]$ ($[n] := \{1, 2, \cdots, n\}$) such that $\sum_{i \in P} a_i = \sum_{j \in [n] \setminus P} a_j$

2 **Subset Sum**: Given some integer t, determine whether there is a subset $P \subseteq [n]$ such that $\sum_{i \in P} a_i = t$

3 **Knapsack**: Given some set of items each with weight w_i and value v_i, and fixed numbers W and V, determine whether there is some subset $P \subseteq [n]$ such that $\sum_{i \in P} w_i \leq W$ and $\sum_{i \in P} v_i \geq V$

For each of the following clearly describe your reduction and justify its correctness.

(a) Find a linear time reduction from Subset Sum to Partition.

(b) Find a linear time reduction from Subset Sum to Knapsack.