CS 170 HW 9

Due **2020-11-02, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 How to Gamble With Little Regret

Suppose that you are gambling at a casino. Every day you play at a slot machine, and your goal is to minimize your losses. We model this as the experts problem. Every day you must take the advice of one of n experts (i.e. play at a slot machine). At the end of each day t, if you take advice from expert i, the advice costs you some c_i^t in $[0, 1]$. You want to minimize the regret R, defined as:

$$R = \frac{1}{T}\left(\sum_{t=1}^{T} c_{i(t)}^t - \min_{1\le i\le n} \sum_{t=1}^{T} c_i^t\right)$$

where $i(t)$ is the expert you choose on day t. Notice that in this definition, you are comparing your losses to the best expert, rather than the best overall strategy.

Your strategy will be probabilities where p_i^t denotes the probability with which you choose expert i on day t. Assume an all powerful adversary (i.e. the casino) can look at your strategy ahead of time and decide the costs associated with each expert on each day. Let C denote the set of costs for all experts and all days. Compute $\max_C(\mathbb{E}[R])$, or the maximum possible (expected) regret that the adversary can guarantee, for each of the following strategies.

(a) Any deterministic strategy, i.e. for each t, there exists some i such that $p_i^t = 1$.

(b) Always choose an expert according to some fixed probability distribution at every time step. That is, fix some $p_1, \ldots, p_n$, and for all t, set $p_i^t = p_i$.

What distribution minimizes the regret of this strategy? In other words, what is $\text{argmin}_{p_1,\ldots,p_n} \max_C(\mathbb{E}[R])$?

This analysis should conclude that a good strategy for the problem must necessarily be randomized and adaptive.

Solution:

(a) $\frac{n-1}{n}$. Consider the case where the cost of the chosen expert is always 1, and the cost of each other expert is 0. Let k be the least-frequently chosen expert, and let m_k be the number of times that expert is chosen. This will result in a regret of $\frac{1}{T}(T - m_k)$

Since the best expert is the one that is chosen least often, the best strategy will try to maximize the number of times we choose the expert that is chosen least often. This means we want to choose all the experts equally many times, so expert k is chosen in at most T/n of the rounds. Therefore, $m_k \leq \frac{T}{n}$, thus the regret is at least $\frac{1}{T}(T - \frac{T}{n}) = \frac{n-1}{n}$.

(Note here that even if our strategy is adaptive, i.e. it chooses an expert on day i based on the losses from days 1 to $i-1$, rather than committing to an expert for day i before seeing the loss for day 1, it still can't achieve regret better than $\frac{n-1}{n}$.)

(b) $(1 - \min_i p_i)$. Like in part (a), the distribution is fixed across all days, so we know ahead of time which expert will be chosen least often in expectation. Let $k = \text{argmin}_i p_i$ be the expert with least cost. Let $c_k^t = 0$ for all t, and let $c_i^t = 1$ for all $i \neq k$ and for all t. This way, $\mathbb{E}\left[\sum_{t=1}^T c_{i(t)}^t\right] = T\left(\sum_{i \neq k} p_i \cdot 1 + p_k \cdot 0\right) = T(1 - p_k)$. We also have $\min_i \sum_{t=1}^T c_i^t = 0$, so we end up with an expected regret of $\frac{1}{T}(T(1-p_k) - 0) = 1 - p_k$.

To minimize the expectation of R is the same as maximizing $\min_i p_i$, which is achieved by the uniform distribution. This gives us regret $\frac{n-1}{n}$ (this is the same worst case regret as in part (b)).

3 Variants on the Experts Problem

Consider the experts problem: We have n experts who predict it will either rain or shine tomorrow, one of which is always correct. Every day we guess based on the experts if it will rain or shine tomorrow. Our goal is to make as few mistakes in our predictions as possible.

(a) Suppose we always guess the prediction of the majority of experts who have been correct so far, breaking ties arbitrarily (Note that this set is always non-empty since one expert is always correct). Show that we make at most $\log n$ mistakes using this strategy.

(b) Suppose there are now k experts who are always correct instead of just one. Give a better upper bound on the number of mistakes the algorithm from the previous part makes makes.

(c) Suppose instead of one expert always being correct, we are only guaranteed that there is one expert who makes at most k mistakes. Consider the following algorithm: Let m be the least mistakes made by any expert so far. Use the prediction of the majority of experts who have made at most m mistakes so far.

Give an upper bound on the number of mistakes made by this algorithm.

Solution:

(a) Every time we make a mistake, at least half of the experts who have been correct so far must also make a mistake, so the set of experts who have not made a mistake decreases by at least half. This set starts at size n, and can't decrease past size 1, so we must make at most $\log n$ mistakes.

(b) Every time we make a mistake, the number of experts we stop listening to decreases in size by at least 1/2. So we go from n experts to at most $2k-1$ experts in at most $\lceil \log(n/(2k-1)) \rceil$ mistakes. At this point, the experts who are always correct are the majority, so we never make a mistake.

(We will give full credit for $\log(n/k)$, since this is the best upper bound you can get without using floor/ceiling functions; this is tight in the case where n, k are both powers of two and half the experts make a mistake every round. Other off-by-one solutions or solutions that are correct up to rounding also get full credit.)

(c) $k(\log n + 1) + \log n$. Every time we make a set of mistakes, the set of experts who have made at most m mistakes must have decreased in size by at least 1/2. When $m < k$, this set goes from size at most n to 0 before m increases by 1, so with every increase in m we make at most $\log n + 1$ mistakes. When $m = k$, this set can go from size n to 1 which takes us at most $\log n$ mistakes.

4 (Optional MT2 Practice) Adding Many Edges At Once

Given an undirected, weighted graph $G(V, E)$, consider the following algorithm to find the minimum spanning tree. This algorithm is similar to Prim's, except rather than grow out a spanning tree from one vertex, it tries to grow out the spanning tree from every vertex at the same time.

procedure FINDMST($G(V, E)$)
  $T \leftarrow \emptyset$
  while T is not a spanning tree **do**
    Let $S_1, S_2 \dots S_k$ be the connected components of the graph with vertices V and edges T
    For each i, let e_i be the minimum-weight edge with exactly one endpoint in S_i
    $T \leftarrow T \cup \{e_1, e_2, \dots e_k\}$
  return T

For example, at the start of the first iteration, we'll have every vertex in its own S_i.

For simplicity, in the following parts you may assume that no two edges in G have the same weight.

(a) Show that this algorithm finds a minimum spanning tree.

(b) Give an exact upper bound (that is, an upper bound without using Big-O notation) on the worst-case number of iterations of the while loop in one run of the algorithm.

(c) Using your answer to the previous part, give an upper bound on the runtime of this algorithm.

Solution:

This algorithm is known as Boruvka's algorithm. Despite perhaps appearing complicated for an MST algorithm, it was discovered in 1926, predating both Prim's and Kruskal's. This algorithm is still of interest due to being easily parallelizable, as well as being the basis for a randomized MST algorithm that runs in expected time $O(|E|)$.

(a) If we add an edge e_i to T, it is because it is the cheapest edge with exactly one endpoint in S_i. So applying the cut property to the cut $(S_i, V - S_i)$ we see that e_i must be in the (unique) minimum spanning tree. So every edge we add must be in the minimum spanning tree, i.e. this algorithm finds exactly the minimum spanning tree.

(b) The number of components in the graph with vertices V and edges T starts out at $|V|$, with one component for each vertex. If there are k components at the start of an iteration, we must add at least $k/2$ edges in that iteration: every component S_i contributes some e_i to the set of edges we're adding, and each edge we add can only have been contributed by up to two components. This means the number of components decreases by at least a factor of 2 in every iteration. Thus, the while loop runs for at most $\log |V|$ iterations.

(c) The previous part shows that at most $\log |V|$ iterations are needed. Each iteration can be performed in $O(|E|)$ time (e.g. you can compute the components in $O(|E|)$ time, and then initialize a table which stores the cheapest edge with exactly one endpoint in each component, and then using a linear scan over all edges fill out this table) giving a runtime bound of $O(|E| \log |V|)$.

5 (Optional MT2 Practice) Huffman Proofs

(a) Prove that in the Huffman coding scheme, if some symbol occurs with frequency more than $\frac{2}{5}$, then there is guaranteed to be a codeword of length 1. Also prove that if all symbols occur with frequency less than $\frac{1}{3}$, then there is guaranteed to be no codeword of length 1.

(b) Suppose that our alphabet consists of n symbols. What is the longest possible encoding of a single symbol under the Huffman code? What set of frequencies yields such an encoding?

Solution:

1. Suppose all codewords have length at least 2 – the tree must have at least 2 levels. Let the weight of a node be the sum of the frequencies of all leaves that can be reached from that node. Suppose the weights of the level-2 nodes are (from left to right for a tree rooted on top) a, b, c, and d. Without loss of generality, assume A and B are joined first, then C and D. So, $a, b \leq c, d \leq a + b$.

   If a or b is greater than $\frac{2}{5}$, then both c and d are greater than $\frac{2}{5}$, so $a+b+c+d > \frac{6}{5} > 1$ (impossible). Now suppose c is greater than $\frac{2}{5}$ (similar idea if d is greater than $\frac{2}{5}$). Then

$a + b > \frac{2}{5}$, so either $a > \frac{1}{5}$ or $b > \frac{1}{5}$ which implies $d > \frac{1}{5}$. We obtain $a + b + c + d > 1$ (impossible).

For the second part suppose there is a codeword of length 1. We have 3 cases. Either the tree will consist of 1 single level-1 leaf node, 2 level-1 leaf nodes, or 1 level-1 leaf node, and 2 level-2 nodes with an arbitrary number of leaves below them in the tree. We will prove the contrapositive of the original statement, that is, that if there is a codeword of length 1, there must be a node with frequency greater than $\frac{1}{3}$

In the first case, our leaf must have frequency 1, so we've immediately found a leaf of frequency more than $\frac{1}{3}$. If the tree has two nodes, one of them has frequency at least $\frac{1}{2}$, so condition is again satsified. In the last case, the tree has one level-1 leaf (weight a), and two level-2 nodes (weights c and d). We have: $b, c \leq a$. So $1 = a + b + c \leq 3a$, or $a \geq \frac{1}{3}$. Again, our condition has been satisfied.

2. The longest codeword can be of length $n - 1$. An encoding of n symbols with $n - 2$ of them having probabilities $1/2, 1/4, \ldots, 1/2^{n-2}$ and two of them having probability $1/2^{n-1}$ achieves this value. No codeword can ever by longer than length $n - 1$. To see why, we consider a prefix tree of the code. If a codeword has length n or greater, then the prefix tree would have height n or greater, so it would have at least $n + 1$ leaves. Our alphabet is of size n, so the prefix tree has exactly n leaves.

6 (Optional MT2 Practice) Bimetallism

There is a blacksmith who can produce n different alloys, where alloy i sells for p_i dollars per unit. One unit of alloy i takes g_i grams of gold and s_i grams of silver to produce. The blacksmith has a total of G grams of gold and S grams of silver to work with, and can produce as many units of each type of alloy as they want within the material constraints. The blacksmith is allowed to produce and sell a non-integer number of units of each alloy.

1. Formulate the linear program to maximize the revenue of the blacksmith, and explain your decision variables, objective function, and constraints.

2. Formulate the dual of the linear program from part (a), and explain your decision variables, objective function, and constraints. The explanations provide economic intuition behind the dual. **Hint:** Formulate the dual first, then think about it from the perspective of the blacksmith when negotiating prices for buying G grams of gold and S grams of silver if they had already signed a contract for the prices for the output alloys p_i. Think about the breakeven point, from which the blacksmith's operations begin to become profitable for at least one alloy.

Solution:

(a) The decision variables x_i correspond to the number of units of each alloy created.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} x_i p_i && \text{(Maximize revenue)} \\ & \sum_{i=1}^{n} x_i g_i \leq G && \text{(Use at most } G \text{ grams of gold)} \\ & \sum_{i=1}^{n} x_i s_i \leq S && \text{(Use at most } S \text{ grams of silver)} \\ x_i \geq 0 \quad & \forall i \in [1...n] && \text{(Cannot produce negative amounts of metal)} \end{aligned}$$

(b) The decision variables y_G, y_S correspond to the prices of the means of production: gold, and silver. The dual poses the following question: if the prices of gold and silver were originally too high for the blacksmith's operations to be profitable, how low can they drop before breaking even? The solution returns the breakeven point; lower prices would finally allow the blacksmith to become profitable.

$$\begin{aligned} \min \quad & G y_G + S y_S && \text{(Minimize total cost of materials)} \\ g_i y_G + s_i y_S \geq p_i \quad & \forall i \in [1...n] && \text{(Cost for producing mixture } i \text{ is at least } p_i) \\ & y_G, y_S \geq 0 && \text{(Cannot set negative prices)} \end{aligned}$$

7 (Optional MT2 Practice) Max Flow Short Answer

(a) True or false: The following problem is equivalent to max-flow. Rather than try to push as much flow as possible from s to t, our goal is to find the largest c such that we can divide all the capacities by c and still be able to route 1 unit of flow from s to t. Justify your answer.

(b) We are given a graph where all edges have infinite capacity and a length ℓ_e. Our goal is to route one unit of flow from s to t such that $\sum_e \ell_e f_e$ is minimized. True or false: The smallest value of $\sum_e \ell_e f_e$ we can achieve is the length of the shortest path. Justify your answer.

(c) Suppose we didn't use a residual graph in max-flow, and instead just used the following algorithm: While there is a path from s to t in the original network such that every edge is not at capacity, push as much flow as possible through any such path.

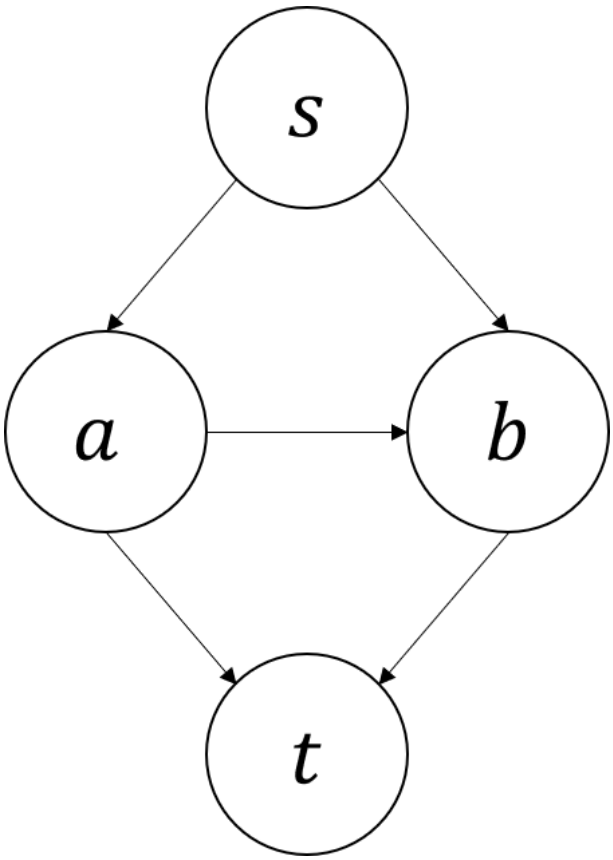
Give a small example of a graph with unit capacities where this algorithm may only find a flow of size 1, but the max flow has size at least 2.

(d) Ford-Fulkerson needs at most F iterations to finish, where F is the size of the max-flow. Give an example of a graph where Ford-Fulkerson could take F iterations, but choosing the right path to push flow on causes it to finish in 2 iterations.

Solution:

(a) True: Suppose the max-flow value is F. Then the largest c we can choose is F.

(b) True: Any flow can be decomposed into a sum of path flows, and using this decomposition $\sum_e \ell_e f_e$ is equal $\sum_p \ell_p f_p$, where f_p is the amount of flow being pushed on the path p and ℓ_p is the length of the path p. This sum is minimized by setting $f_e = 1$ for the path p with the smallset length ℓ_p, i.e. the flow that minimizes $\sum_e \ell_e f_e$ only uses a single path. So the problem is equivalent to shortest path.



(c) In this graph, this algorithm could choose the path s, a, b, t and then not be able to choose any other path.

(d) Same as the above graph, but set the capacity of all edges except (a, b) to $F/2$.

A bad choice of paths to alternate between pushing flow on s, a, b, t and s, b, a, t, only pushing 1 unit of flow at a time.

A good choice of paths is to push flow on s, a, t and then s, b, t.

8 (Optional MT2 Practice) 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?

(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.

(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.

Solution:

(a) To get the column player's payoff matrix, we negate the payoff matrix and take its transpose. So we get that the row and column players' payoff matrices are the same matrix. In turn, they must have the same expected payoff, but also the sum of their expected payoffs must be 0, so both players must have expected payoff 0.

(b) False: Consider the 2-by-3 payoff matrix:

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \end{bmatrix}$$

The row player's optimal strategy is to choose the two rows with equal probability - note that the column player doesn't care about choosing column 1 vs column 3, so this game is no different than the zero-sum game for the 2-by-2 payoff matrix: $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$

(c) True: Consider the optimal strategy for the row player. If the row player chooses rows 1, 2, 3 with probabilities p_1, p_2, p_3, they can instead choose row 1 with probability 0, row 2 with probability $p_2 + p_1/2$, and row 3 with probability $p_3 + p_1/2$. The expected payoff of this strategy is the same, so this strategy is also optimal.