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.

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.

$$M = \begin{bmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,m} \\ M_{2,1} & M_{2,2} & \cdots & M_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n,1} & M_{n,2} & \cdots & M_{n,m} \end{bmatrix}$$

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:

$$\max \ z$$
$$M_{1,1} \cdot p_1 + \cdots + M_{n,1} \cdot p_n \geq z$$
$$M_{1,2} \cdot p_1 + \cdots + M_{n,2} \cdot p_n \geq z$$
$$\vdots$$
$$M_{1,m} \cdot p_1 + \cdots + M_{n,m} \cdot p_n \geq z$$
$$p_1 + p_2 + \cdots + p_n = 1$$
$$p_1, p_2, \ldots, p_n \geq 0$$

The dual represents fixing the minimizer's choices to a probabilistic distribution. If we let the probability that the minimizer chooses choice j be q_j, then the dual is the following:

$$\min \ w$$
$$M_{1,1} \cdot q_1 + \cdots + M_{1,m} \cdot q_m \leq w$$
$$M_{2,1} \cdot q_1 + \cdots + M_{2,m} \cdot q_m \leq w$$
$$\vdots$$
$$M_{n,1} \cdot q_1 + \cdots + M_{n,m} \cdot q_m \leq w$$
$$q_1 + q_2 + \cdots + q_m = 1$$
$$q_1, q_2, \ldots, q_m \geq 0$$

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.

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?

(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.

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 $m_1, m_2, \ldots m_n$ in some order. For example, the payoff matrix might look like:

$$M = \begin{bmatrix} m_1 & m_2 & m_3 \\ m_2 & m_3 & m_1 \\ m_3 & m_1 & m_2 \end{bmatrix}$$

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).

Solution: Both players' optimal strategy is to choose a row/column uniformly at random. The expected payoff of this strategy is $\sum_k m_k/n$.

To show these are optimal, note that if the column player picks this strategy, any strategy the row player chooses has expected payoff $\sum_k m_k/n$. By symmetry, this also implies that if the row player picks this strategy, any strategy the column player picks achieves the same expected payoff. By duality, this must be the optimal pair of mixed equilibrium strategies.

3 Zero-Sum Games

Alice and Bob are playing a zero-sum game whose payoff matrix is shown below. The ij^{th} entry of the matrix shows the payoff that Alice receives if she plays strategy i and Bob plays strategy j. Alice is the row player and is trying to maximize her payoff, and Bob is the column player trying to minimize Alice's payoff.

Alice \Bob	1	2
1	4	1
2	2	5

Now we will write a linear program to find a strategy that maximizes Alice's payoff. Let the variables of the linear program be x_1, x_2 and p, where x_i is the probability that Alice plays row i and p denotes Alice's payoff.

(a) Write the linear program for choosing Alice's strategy to maximize her payoff.

(b) Write a linear program from Bob's perspective trying to minimizing Alice's payoff. Let the variables of the linear program be y_1, y_2 and p, where y_i is the probability that Bob plays strategy i and p denotes Alice's payoff.

(c) As covered in lecture, Bob's linear program and Alice's are dual to each other. How can you see that this is the case for the LPs you have written here? Either take the dual mechanically or make a concise argument. (Hint: You may want to transform the linear programs into a form where it is easy to take the dual, if they are not already in such a form).

(d) What is the optimal solution and what is the value of the game?

Solution:

(a)

$$\begin{aligned} \max\, & p \\ & p \leq 4x_1 + 2x_2 \\ & p \leq x_1 + 5x_2 \\ & x_i \geq 0 \\ x_1 + x_2 &= 1 \end{aligned}$$

(b)

$$\begin{aligned} \min\, & p \\ & p \geq 4y_1 + y_2 \\ & p \geq 2y_1 + 5y_2 \\ & y_i \geq 0 \\ y_1 + y_2 &= 1 \end{aligned}$$

(c) To make our lives easier, we will make a slight adjustment to Alice's linear program: we will replace $x_1 + x_2 = 1$ with $x_1 + x_2 \leq 1$. This works because decreasing x_1 and x_2 can only decrease p, so our new constraint will hold with equality at the optimum. We will also write Alice's linear program in a more manageable form where all variables appear (possibly with zero coefficients) in the objective function and all constraints, and the right side of every constraint is a constant, like so:

$$\begin{aligned} \max\, & 0x_1 + 0x_2 + 1p \\ & -4x_1 - 2x_2 + 1p \leq 0 \\ & -1x_1 - 5x_2 + 1p \leq 0 \\ & 1x_1 + 1x_2 + 1p \leq 1 \\ & x_i \geq 0 \end{aligned}$$

With Alice's linear program in this more familiar form, we can now take the dual mechanically. We will call our dual variables y_1, y_2, and p:

$$\begin{aligned} \min\, & 0y_1 + 0y_2 + 1p \\ & -4y_1 - 1y_2 + 1p \geq 0 \\ & -2y_1 - 5x_2 + 1p \geq 0 \\ & 1y_1 + 1y_2 + 0p \geq 1 \\ & y_i \geq 0 \end{aligned}$$

Applying the same transformations to Bob's linear program as we did to Alice's, we can see that it is exactly equal to the program we have just found. Alice's and Bob's linear programs are dual to each other. This should make some sense: Once both players have announced probabilistic strategies, the expected payoff of the game is fixed. The action of whichever player goes first is to place a bound (in Alice's case, the maximum possible lower bound, and in Bob's case, the minimum possible upper bound) on the payoff achievable by the other player.

(d)

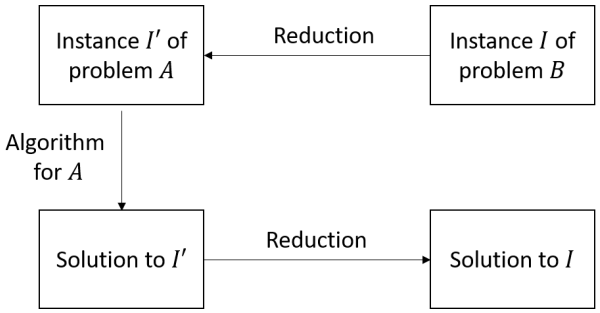
$$x_1 = \frac{1}{2}$$
$$x_2 = \frac{1}{2}$$
$$p = 3$$

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 **both** (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.

4 Bad Reductions

In each part we make a wrong claim about some reduction. Explain for each one why the claim is wrong.

(a) The shortest simple path problem with non-negative edge weights can be reduced to the longest simple path problem by just negating the weights of all edges. There is an efficient algorithm for the shortest simple path problem with non-negative edge weights, so there is also an efficient algorithm for the longest path problem.

(b) We have a reduction from problem B to problem A that takes an instance of B of size n, and creates a corresponding instance of A of size n^2. There is an algorithm that solves A in quadratic time. So our reduction also gives an algorithm that solves B in quadratic time.

(c) We have a reduction from problem B to problem A that takes an instance of B of size n, and creates a corresponding instance of A of size n in $O(n^2)$ time. There is an algorithm that solves A in linear time. So our reduction also gives an algorithm that solves B in linear time.

(d) Minimum vertex cover can be reduced to shortest path in the following way: Given a graph G, if the minimum vertex cover in G has size k, we can create a new graph G' where the shortest path from s to t in G' has length k. The shortest path length in G' and size of the minimum vertex cover in G are the same, so if we have an efficient algorithm for shortest path, we also have one for vertex cover.

Solution:

(a) The reduction is in the wrong direction. By reducing shortest simple path to longest simple path, we showed that longest simple path is at least as hard as shortest simple path, but the longest simple path problem could be much harder (indeed, we don't know of any algorithm taking less than exponential time).

(b) The reduction blows up the size of the instance, so running an algorithm on the instance of A would actually take quartic/$O(n^4)$ time, not quadratic time. Furthermore, the runtime of the reduction step is polynomial in n, so it is possible for it to take $O(n^d)$ time for some $d > 2$, making the overall algorithm slower than quadratic time even before solving the instance of A.

(c) Now the reduction doesn't blow up the size of the instance, but it takes $O(n^2)$ time, so combining the reduction and algorithm for A only gives an $O(n^2)$ time algorithm.

(d) This reduction needs an algorithm for minimum vertex cover to compute what k is, which defeats the point of the reduction. Effectively, this just shows that minimum vertex cover is at least as hard as itself.

5 Graph Coloring Problem

An undirected graph $G = (V, E)$ is k-colorable if we can assign every vertex a color from the set $1, \cdots, k$, such that no two adjacent vertices have the same color. In the k-coloring problem, we are given a graph G and want to output "Yes" if it is k-colorable and "No" otherwise.

(a) Show how to reduce the 2-coloring problem to the 3-coloring problem. That is, describe an algorithm that takes a graph G and outputs a graph G', such that G' is 3-colorable if and only if G is 2-colorable. To prove the correctness of your algorithm, describe how to construct a 3-coloring of G' from a 2-coloring of G and vice-versa. (No runtime analysis needed).

(b) The 2-coloring problem has a $O(|V| + |E|)$-time algorithm. Does the above reduction imply an efficient algorithm for the 3-coloring problem? If yes, what is the runtime of the resulting algorithm? If no, justify your answer.

Solution:

(a) To construct G' from G, add a new vertex v^* to G, and connect v^* to all other vertices.

If G is 2-colorable, one can take a 2-coloring of G and assign color 3 to v^* to get a 3-coloring of G'. Hence G' is 3-colorable.

If G' is 3-colorable, since v^* is adjacent to every other vertex in the graph, in any valid 3-coloring v^* must be the only vertex of its color. This means all remaining vertices only use 2 colors total, i.e. G is 2-colorable. We can recover the 2-coloring of G from the 3-coloring of G' by just using the 3-coloring of G', ignoring v^* (and remapping the colors in the 3-coloring except for v^*'s color to colors 1 and 2).

(b) No, the reduction is in the wrong direction. This reduction shows 3-coloring is at least as hard as 2-coloring, but it could be much harder.

(Comment: Indeed, it is suspected that there is no 3-coloring algorithm running in time $O(2^{|V|^{.99}})$. This is part of a common and somewhat mystical trend we will see more examples of very soon: lots of problems go from easy to hard when we change a 2 to a 3 in the problem description.)

6 Optimization versus Search

Recall the following definition of the Traveling Salesman Problem, which we will call Search-tsp. We are given a complete graph G of whose edges are weighted and a budget b. We want to find a tour (i.e., path) which passes through all the nodes of G and has length $\leq b$, if such a tour exists.

The optimization version of this problem (which we call Min-tsp) asks directly for the shortest tour.

(a) Show that if Search-tsp can be solved in polynomial time, then so can Min-tsp.

(b) Do the reverse of (a), namely, show that if Min-tsp can be solved in polynomial time, then so can Search-tsp.

Solution:

(a) Do a binary search over all possible lengths of the optimal tour, going from 0 to the sum of all distances. Note that binary search is necessary here and we can't just increment the value of b by 1 each time since the sum of all distance is exponential in the size of the input.

(b) Run tsp-opt and return the optimal value. If it is greater than b, then no solution exists, by the definition of optimality.