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 the 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:

$$\max \; z$$

$$M_{1,1} \cdot p_1 + \cdots + M_{n,1} \cdot p_n \geq z$$

$$M_{1,2} \cdot p_1 + \cdots + M_{n,2} \cdot p_n \geq z$$

$$\vdots$$

$$M_{1,m} \cdot p_1 + \cdots + M_{n,m} \cdot p_n \geq z$$

$$p_1 + p_2 + \cdots + p_n = 1$$

$$p_1, p_2, \ldots, p_n \geq 0$$

or in other words,

$$\max z \text{ s.t. } M^\top p \geq z\mathbf{1}, \; \mathbf{1}^\top p = 1, \; p \geq 0$$

where $p = (p_1, \ldots, p_n)$.

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:

$$\min \; w$$

$$M_{1,1} \cdot q_1 + \cdots + M_{1,m} \cdot q_m \leq w$$

$$M_{2,1} \cdot q_1 + \cdots + M_{2,m} \cdot q_m \leq w$$

$$\vdots$$

$$M_{n,1} \cdot q_1 + \cdots + M_{n,m} \cdot q_m \leq w$$

$$q_1 + q_2 + \cdots + q_m = 1$$

$$q_1, q_2, \ldots, q_m \geq 0$$

or in other words,

$$\min w \text{ s.t. } Mq \leq w\mathbf{1},\ \mathbf{1}^\top q = 1,\ q \geq 0$$

where $q = (q_1, \ldots, q_m)$.

By strong duality, the optimal value of the game is the same regardless of whether you fix the minimizer's distribution first or the maximizer's distribution first; i.e. $z^* = w^*$, *or* $z^* + (-w^*) = 0$.

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?

Hint: try rewriting the minimizer's (i.e. column player's) LP as a maximization problem. Then, use the definition of a ZSG (i.e. when both players try to maximize their payoff, their optimal payoffs sum to 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.

(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. Justify your answer.

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).

Note: some of this content below is a preview of what we'll be seeing over the next few weeks! We'll see it more in lecture soon.

If there exists a polynomial-time reduction from problem A to problem B, problem B is at least as hard as problem A. From this, we can define complexity class which sort of gauge 'hardness'.

Complexity Class Definitions

- NP: a problem in which a potential solution can be verified in polynomial time.
- P: a problem which can be solved in polynomial time.
- NP-Complete: a problem in NP which all problems in NP can polynomial-time reduce to.
- NP-Hard: any problem which is at least as hard as an NP-Complete problem.

Prove a problem is NP-Complete
To prove a problem is NP-Complete, you must prove the problem is in NP and it is in NP-Hard.

To prove that a problem is in NP, you must show there exists a polynomial verifier for it.

To prove that a problem is **NP-hard,** you can reduce an NP-Complete problem to your problem.

3 NP or not NP, that is the question

For the following questions, circle the (unique) condition that would make the statement true.

Note: for now, you may not be able to solve all of these questions formally. Try to solve these intuitively! Revisit these problems after Thursday's lecture and try to solve them more formally then.

(a) Minimum Spanning Tree is in NP.

Always True   True iff $\mathsf{P} = \mathsf{NP}$ True iff $\mathsf{P} \neq \mathsf{NP}$ Always False

(b) 2-SAT is NP-complete. (Informally, 2-SAT is one of the hardest problems in NP)

Always True   True iff $\mathsf{P} = \mathsf{NP}$ True iff $\mathsf{P} \neq \mathsf{NP}$ Always False

(c) If B is in NP, then for any problem $A \in \mathsf{P}$, there exists a polynomial-time reduction from A to B. (Informally, B is at least as hard as A)

Always True   True iff $\mathsf{P} = \mathsf{NP}$ True iff $\mathsf{P} \neq \mathsf{NP}$ Always False

(d) If B is NP-complete, then for any problem $A \in \mathsf{NP}$, there exists a polynomial-time reduction from A to B. (Informally, B is at least as hard as A)

Always True   True iff $\mathsf{P} = \mathsf{NP}$ True iff $\mathsf{P} \neq \mathsf{NP}$ Always False

4 Pseudo-polynomial Time Algorithms

Recall that a pseudo-polynomial time algorithm is one where the runtime is polynomial in the numerical values of the inputs but not the length of the input (i.e. the number of bits used to represent the input). In this problem, we gain a little more intuition for pseudo-polynomial time algorithms and why we distinguish them from algorithms in P.

(a) Consider a graph algorithm, i.e. Bellman-Ford, where the input bounds are generally $|V|$ and $|E|$. Let's ignore the edge count for now. How many undirected graphs are there with $|V| = 10$ vertices? Put another way, how many different problems do we have to distinguish between and be able to solve in $O(10^3)$ time (or better) without exception? If we double the value of $|V|$, how does the number of possible graphs scale?

(b) For the knapsack problem, our bounds are in terms of the number of items as well as the total storage weight capacity of our knapsack, i.e. $O(nW)$. Let us consider one item i. How many possibilities are there for the weight of item i if $W = 10$ (such that it can possibly be part of an optimal solution)? If we double the value of W, what how does the number of weight possibilities scale? How does this compare to the previous part?

(c) Explain why the following algorithm for testing whether a number is prime is a pseudo-polynomial time algorithm in n assuming constant arithmetic: Given an integer n: test all numbers i from 2 to $n-1$ to determine whether i divides n. If it does, then output FALSE. Otherwise, output TRUE. *Remark: the same logic should hold if you only test numbers i up to $\sqrt{n}$.*