CS 170 Discussion 9 Reference Sheet

Canonical Form. The canonical form of a linear program is

$$\begin{aligned} &\text{maximize } c^\top x \\ &\text{subject to } Ax \leq b \\ &\qquad\qquad x \geq 0 \end{aligned}$$

where $x \geq 0$ means that every entry of the vector x is greater than or equal to 0. Here, $f(x) = c^\top x$ is the **objective function**, and $Ax \leq b, x \geq 0$ are the **constraints**. The **feasible region** is the intersection of all the halfspaces defined by the constraints.

Dual. The dual of the canonical LP is

$$\begin{aligned} &\text{minimize } y^\top b \\ &\text{subject to } y^\top A \geq c^\top \\ &\qquad\qquad y \geq 0 \end{aligned}$$

Weak duality: The objective value of any feasible primal $\leq$ objective value of any feasible dual. In other words, if we define x^* to be the optimizer of the LP above (in canonical form) and y^* to be the optimizer of its dual, weak duality implies that

$$c^\top x^* \leq (y^*)^\top b$$

Strong duality: The *optimal* objective values of a primal LP and its dual are equal. In other words,

$$c^\top x^* = (y^*)^\top b$$

Note that strong duality always holds for linear programs as long as the primal or dual are feasible.

Simplex:

1. Start at an arbitrary vertex.
2. Denote the current vertex as x.
3. Look at all neighboring vertices y to x.
4. Find the neighbor y^* with the best objective value (if the LP is minimizing, y^* should have the smallest objective value, etc.).
5. If y^* has a better value than x, then we move to (i.e. visit) y^* and repeat steps 2 to 4. Otherwise, we have found the optimizer of the LP, and simply return x.

$\hookrightarrow$ Runtime: worst case exponential time, average case polynomial time.

LP Solver Runtime: Any LP can be solved in (worst case) polynomial time using the Ellipsoid or Interior Point Method (IPM). *Note: you do not need to know how the Ellipsoid method or IPM work, but you should know that they can be used to solve an LP in polynomial time.*

Zero Sum Games: In this game, there are two players: a maximizer and a minimizer. We generally write the payoff matrix M from 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 minimize 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:

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

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:

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

or in other words,

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

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

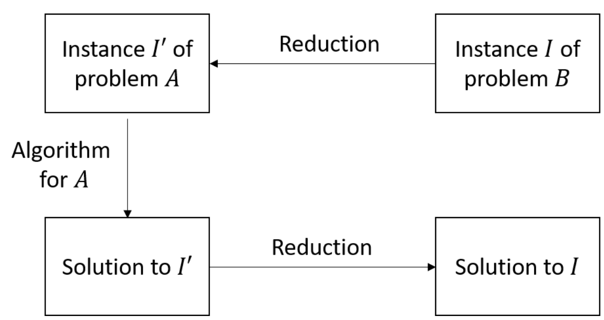
By strong duality, the optimal value of the game is the same regardless of whether you fix the minimizer's distribution first or the maximizer's distribution first; i.e. $z^* = w^*$, *or* $z^* + (-w^*) = 0$.

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 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.

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, an 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 instance I' of problem A has a solution, then so does instance I of problem B.

(2) If instance I of B has a solution, then so does instance I' of problem A.

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.