1 Primals and Duals: Zero Sum Games

We can represent various situations of conflict in life in terms of matrix games. For example, the game shown below is the rock-paper-scissors game. The Row player chooses a row strategy, the Column player chooses a column strategy, and then Column pays to Row the value at the intersection (if it is negative, Row ends up paying Column).

\[
\begin{pmatrix}
  r & 0 & -1 & 1 \\
  p & 1 & 0 & -1 \\
  s & -1 & 1 & 0 \\
\end{pmatrix}
\]

Games do not necessarily have to be symmetric (that is, Row and Column have the same strategies, or, in terms of matrices, \(A = -AT\)). For example, in the following fictitious Clinton-Dole game the strategies may be the issues on which a candidate for office may focus (the initials stand for “economy,” “society,” “morality,” and “tax-cut”) and the entries are the number of voters lost by Column.

\[
\begin{pmatrix}
m & t \\
e & 3 & -1 \\
s & -2 & 1 \\
\end{pmatrix}
\]

We want to explore how the two players may play “optimally” these games. It is not clear what this means. For example, in the first game there is no such thing as an optimal “pure” strategy (it very much depends on what your opponent does; similarly in the second game). But suppose that you play this game repeatedly. Then it makes sense to randomize. That is, consider a game given by an \(m \times n\) matrix \(G_{ij}\); define a mixed strategy for the row player to be a vector \((x_1, \ldots, x_m)\), such that \(x_i \geq 0\), and \(\sum_{i=1}^m x_i = 1\). Intuitively, \(x_i\) is the probability with which Row plays strategy \(i\). Similarly, a mixed strategy for Column is a vector \((y_1, \ldots, y_n)\), such that \(y_j \geq 0\), and \(\sum_{j=1}^n y_j = 1\).

Suppose that, in the Clinton-Dole game, Row decides to play the mixed strategy \((.5, .5)\). What should Column do? The answer is easy: if the Row player’s \(x_i\)’s are given, there is a pure strategy (that is, a mixed strategy with only one \(y_j \neq 0\) ) that is optimal. We can find it by calculating the expected winnings \(\sum_{i=1}^m G_{ij}x_i\) for each column \(j\). In the Clinton-Dole game, Column has an expected win value of 0.5 for \(m\) and 0 for \(t\), so Column should pick \(m\).

Thus, if Column knew Row’s mixed strategy, s/he would end up paying the smallest among the \(n\) outcomes \(\sum_{i=1}^m G_{ij}x_i\), for \(j = 1, \ldots, n\), or the value \(\min_j \sum_{i=1}^m G_{ij}x_i\). On the other hand, Row will seek the mixed strategy that maximizes this minimum:

\[
\max_{\vec{x}} \min_j \sum_{i=1}^m G_{ij}x_i.
\]

This value would be the best possible guarantee about an expected outcome that Row can have by choosing a mixed strategy. Let us call this guarantee \(z\); what Row is trying to do is solve the
following LP:

\[
\begin{align*}
\text{max } z \\
z & \leq -3x_1 + 2x_2 \\
z & \leq x_1 - x_2 \\
x_1 + x_2 & = 1 \\
x_1, x_2 & \geq 0
\end{align*}
\]

Here, by manipulating \(x_1\) and \(x_2\), Row is trying to maximize \(z\) subject to the constraint that \(z\) is at most the expected winnings for any choice that Column makes. Here, if Column chooses \(m\), then we must have \(z \leq 3x_1 - 2x_2\). If Column chooses \(t\), then \(z \leq -x_1 + x_2\).

**Exercise 1.** Write the LP that Column would try to solve.

The crucial observation now is that *these LP’s are dual to each other*, and hence have the same optimum \(V\)!

Let us summarize: By solving an LP, Row can guarantee an expected income of at least \(V\), and by solving the dual LP, Column can guarantee an expected loss of at most the same value. It follows that this is the uniquely defined optimal play; note that we weren’t *a priori* certain that such a play exists! \(V\) is called the *value of the game*. In this case, the optimum mixed strategy for Row is \((3/7, 4/7)\), and for Column \((2/7, 5/7)\), with a value of \(1/7\) for the Row player.

The existence of mixed strategies that are optimal for both players and achieve the same value is a fundamental result in Game Theory called the *min-max theorem*. It can be written in equations as follows:

\[
\max_x \min_y \sum x_i y_j G_{ij} = \min_y \max_x \sum x_i y_j G_{ij}.
\]

It is surprising, because the left-hand side, in which Column optimizes last, and therefore has presumably an advantage, should be intuitively smaller than the right-hand side, in which Column decides first. Duality equalizes the two, as it does in max-flow min-cut.

## 2 P vs NP

### 2.1 Definitions

There are two classes of problems that we will examine in this course: P and NP. In today’s section, we will review the difference between the two, as well as get some more practice classifying something as P, NP, and NP-Complete.

- **P** - the set of all problems with yes/no answers that can be *answered* in polynomial time (i.e. a solution can be found in \(O(n^k)\) steps for some non-negative integer \(k\)).
- **NP** - the set of all problems with yes/no answers that can be *verified* in polynomial time using a “certificate” (i.e. we can determine whether the answer is yes or no in \(O(n^k)\) steps using the certificate).
Almost everything we’ve seen in class so far has been a P-class problem (shortest paths, MSTs, dynamic programming, max flow, etc.). Examples of NP-class problems are 3-SAT, integer linear programming, vertex cover, max clique, and more.  Note that P is a subclass of NP; any correct solution we calculate in polynomial time must be verifiable in polynomial time, since it was generated by a correct polynomial-time algorithm.

The following may be a useful characterization: P is the set of problems solvable by polynomial-time deterministic algorithms, while NP is the set of problems solvable by polynomial-time non-deterministic algorithms. Non-deterministic algorithms explore all possibilities at once (e.g. exploring all children of a node at once). Think about why this characterization works!

### 2.2 Reductions

A reduction is a procedure $R$ which transforms an input for problem A into an input for problem B. In a reduction, we should think of problems in the following manner: given an input (which serves to “configure” the problem), is it possible to find a solution? As an example: the 3-SAT problem takes an expression in the form of clauses and asks “is it possible to satisfy this expression?” This transformation must maintain the integrity of the answer; that is, the answer to A for an input $x$ is the same as the answer to B for the input $R(x)$. If we are able to find such a procedure $R(x)$, then we can solve A by using B. We can conclude that B is at least as "hard" to solve as A.

Typically, if we want to show that a problem A is easy (e.g. it’s a P-class problem), we reduce A to something else that we know is easy, since then A can be no harder than that problem. If we want to show that A is a hard problem to solve (e.g. it’s a NP-class problem), we usually reduce a known, difficult problem to A; then A is at least as hard as that problem.

### 2.3 NP-Complete

NP-complete problems are the "hardest" problems in NP; all other problems in NP reduce to them. To show that a problem A is NP-complete, we reduce a known NP-complete problem to A. We discovered NP-complete problems sequentially in the following way:

- **Circuit-SAT**: the first NP-complete problem, introduced by Cook’s Theorem (see lecture).
- **3-SAT**: Circuit-SAT reduced to this by replacing gates with (complex) boolean expressions.
- **Integer Linear Programming**: 3-SAT reduced to this by replacing each literal with $x$ or $1 - x$ and constraining the sum of each clause to be greater than 1.
- **Independent Set**: 3-SAT reduced to this by constructing a strange graph where vertices represented possible assignments of the literals, and edges connected conflicting assignments.
- **Vertex Cover**: Independent Set and Vertex Cover reduce to each other by observing that $C$ is an independent set within $V$ if and only if $V - C$ is a vertex cover.

You may freely use the fact that these problems are NP-complete in your proofs.
3 Reduction Practice Problems

Exercise 2. Show that the following optimization problem is in P:

Call two paths \textit{edge-disjoint} if they have no edges in common. Given a directed graph $G = (V, E)$ and two nodes $s, t \in V$, find the maximum number of edge-disjoint paths from $s$ to $t$.

Exercise 3. Recall the Set Cover problem: given a set $U$ of elements and set of $m$ subsets $S = \{S_1, S_2, \cdots, S_m\}$ with $S_i \subseteq U$ for all $i$, is there a collection of at most $k$ of these subsets whose union equals $U$? You may remember that there is a greedy algorithm which is off by a factor of $O(\log n)$. Show that Set Cover is actually NP-Complete.
Exercise 4. A 3-coloring of a graph $G = (V, E)$ is a assignment of colors to the vertices $f : V \rightarrow \{\text{red, green, blue}\}$ such that for every edge $(u, v) \in E$, $f(u) \neq f(v)$. Show that 3-coloring is NP-complete.

Hint: Consider the following graph with the clause $(x_1 \lor \bar{x}_2 \lor \bar{x}_3)$: