CS 580: Algorithm Design and Analysis

Jeremiah Blocki
Purdue University
Spring 2019

Homework 4: Due tomorrow (March 8) at 11:59 PM
Recap

**Linear Programming**
- Very Powerful Technique (Subject of Entire Courses)
- **Our Focus:** Using Linear Programming as a Tool
  - Solving Network Flow using Linear Programming
  - Finding Minimax Optimal Strategy in 2-Player Zero Sum Game
  - Operations Research (Brewery Example)

**Solving Linear Programs**
- Simplex Intuition:
  - Optimal point is an “extreme point”
  - No “local optimum”
- Simplex Runs in Exponential Time in Worst Case
  - But other algorithms (e.g., Ellipsoid) run in polynomial time
### 2-Player Zero-Sum Games

Example: Shooter-Goalie

<table>
<thead>
<tr>
<th></th>
<th>Block Left</th>
<th>Block Right</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shoot Left</strong></td>
<td>1/2</td>
<td>0.9</td>
</tr>
<tr>
<td><strong>shoot Right</strong></td>
<td>0.8</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Shooter scores 80% of time when shooter aims right and goalie blocks left.

**Minimax Optimal Strategy** (possibly randomized) best strategy you can find given that opponent is rational (and knows your strategy).

How can we find Minimax Optimal Strategy?
2-Player Zero-Sum Games

Player 1 Mixed Strategy: $p_1, \ldots, p_n$ (n actions)

Suppose Player 2 plays action $j$

- Player 1 receives reward $m_{ij}$ with probability $p_i$
- Player 1 receives expected reward $\sum_i p_i m_{ij}$ when player 2 plays $j$

- Player 2 receives expected reward $-\sum_i p_i m_{ij}$

Player 2 will select $j$ to maximize $-\sum_i p_i m_{ij}$
(Equivalently, minimize $\sum_i p_i m_{ij}$)

Therefore, expected value of Strategy $p_1, \ldots, p_n$ to player 1 is

$$V_1(p_1, \ldots, p_n) = \min_j \left\{ \sum_i p_i m_{ij} \right\}$$

Best Strategy $p_1^*, \ldots, p_n^*$ should maximize $V_1^* = V_1(p_1^*, \ldots, p_n^*)$
Finding Minimax Optimal Solution using Linear Programming

Variables: $p_1, \ldots, p_n$ and $v$ ($p_i$ is probability of action $i$)

Goal: Maximize $v$ (our expected reward).

Constraints:

- $p_1, \ldots, p_n \geq 0$
- $1 \geq p_1 + \ldots + p_n \geq 1$
- For all columns $j$ we have $\sum_i p_im_{ij} \geq v$

$m_{ij}$ denotes reward when player 1 takes action $i$ and player 2 takes action $j$.

Expected reward when player 2 takes action $j$

Duality: $V_1^* = -V_2^*$
Example: Shooter-Goalie

<table>
<thead>
<tr>
<th></th>
<th>Block Left</th>
<th>Block Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoot Left</td>
<td>1/2</td>
<td>0.9</td>
</tr>
<tr>
<td>shoot Right</td>
<td>0.8</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Goalie Blocks Left

Goalie Blocks Right

Maximize \[
\begin{cases}
S_L \geq 0, S_R \geq 0, 1 \leq S_L + S_R \leq 1, \\
\frac{1}{2} S_L + \frac{4}{5} S_R \geq v, \\
\frac{9}{10} S_L + \frac{1}{3} S_R \geq v
\end{cases}
\] \{v, S_L, S_R\}
Example: Shooter-Goalie

<table>
<thead>
<tr>
<th>Shoot Left</th>
<th>Block Left</th>
<th>Block Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoot Left</td>
<td>1/2</td>
<td>0.9</td>
</tr>
<tr>
<td>shoot Right</td>
<td>0.8</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Maximize \[
\begin{cases} 
  S_L \geq 0, S_R \geq 0, 1 \leq S_L + S_R \leq 1, \\
  S_L(1/2) + S_R(4/5) \geq v, \\
  S_L(9/10) + S_R(1/3) \geq v 
\end{cases}
, \{v, S_L, S_R\}
\]

Solution: \{v \rightarrow 0.638462, L \rightarrow 0.538462, R \rightarrow 0.461538\}

- Shooter guaranteed to score at least 63.846% of the time by shooting left 53.846% of time
2-Player Zero-Sum Games

**Example: Shooter-Goalie**

<table>
<thead>
<tr>
<th></th>
<th>Block Left</th>
<th>Block Right</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shoot Left</strong></td>
<td>1/2</td>
<td>0.9</td>
</tr>
<tr>
<td><strong>Shoot Right</strong></td>
<td>0.8</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Shooter Minimax (solve in Mathematica)

Maximize

\[
\begin{cases}
B_L \geq 0, B_R \geq 0, & v, \\
B_L \left(-\frac{1}{2}\right) + B_R \left(-\frac{9}{10}\right) \geq v, & 1 \leq B_L + B_R \leq 1, \\
B_L \left(-\frac{4}{5}\right) + B_R \left(-\frac{1}{3}\right) \geq v
\end{cases}
\}

\{v, B_L, B_R\}

Pr block left/right
2-Player Zero-Sum Games

Example: Shooter-Goalie

<table>
<thead>
<tr>
<th></th>
<th>Block Left</th>
<th>Block Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoot Left</td>
<td>1/2</td>
<td>0.9</td>
</tr>
<tr>
<td>shoot Right</td>
<td>0.8</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Maximize
\[
\left\{ \begin{array}{l} v, \\
B_L \geq 0, B_R \geq 0, 1 \leq B_L + B_R \leq 1, \\
B_L(-1/2) + B_R(-9/10) \geq v, \\
B_L(-4/5) + B_R(-1/3) \geq v \end{array} \right\}, \{v, B_L, B_R\}
\]

Solution: \{v \rightarrow -0.638462, B_L \rightarrow 0.653846, B_R \rightarrow 0.346154\}
- Goal scored at most 63.846% if goalie blocks right 65.384% of the time
Chapter 8

NP and Computational Intractability
Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns.

- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.

Ex.

- \( O(n \log n) \) interval scheduling.
- \( O(n \log n) \) FFT.
- \( O(n^2) \) edit distance.
- \( O(n^3) \) bipartite matching.
- Circulation via Network Flow
  - Bipartite Matching via Network Flow
  - Minimax Strategy via Linear Programming

Algorithm design anti-patterns.

- NP-completeness. \( O(n^k) \) algorithm unlikely.
- PSPACE-completeness. \( O(n^k) \) certification algorithm unlikely.
- Undecidability. No algorithm possible.
8.1 Polynomial-Time Reductions
Classify Problems According to Computational Requirements

**Q.** Which problems will we be able to solve in practice?


Those with polynomial-time algorithms.

<table>
<thead>
<tr>
<th>Yes</th>
<th>Probably no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest path</td>
<td>Longest path</td>
</tr>
<tr>
<td>Matching</td>
<td>3D-matching</td>
</tr>
<tr>
<td>Min cut</td>
<td>Max cut</td>
</tr>
<tr>
<td>2-SAT</td>
<td>3-SAT</td>
</tr>
<tr>
<td>Planar 4-color</td>
<td>Planar 3-color</td>
</tr>
<tr>
<td>Bipartite vertex cover</td>
<td>Vertex cover</td>
</tr>
<tr>
<td>Primality testing</td>
<td>Factoring</td>
</tr>
</tbody>
</table>
Classify Problems

Desiderata. Classify problems according to those that can be solved in polynomial-time and those that cannot.

Provably requires exponential-time.
- Given a Turing machine, does it halt in at most k steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

Frustrating news. Huge number of fundamental problems have defied classification for decades.

This chapter. Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.
**Desiderata**. Suppose we could solve $X$ in polynomial-time. What else could we solve in polynomial time?

**Reduction.** Problem $X$ polynomial reduces to problem $Y$ if arbitrary instances of problem $X$ can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem $Y$.

**Notation.** $X \leq_P Y$.

**Remarks.**

- We pay for time to write down instances sent to black box $\Rightarrow$ instances of $Y$ must be of polynomial size.
- Note: Cook reducibility.

Don’t confuse with reduces from computational model supplemented by special piece of hardware that solves instances of $Y$ in a single step in contrast to Karp reductions.
Polynomial-Time Reduction

**Purpose.** Classify problems according to relative difficulty.

**Design algorithms.** If $X \leq_p Y$ and $Y$ can be solved in polynomial-time, then $X$ can also be solved in polynomial time.

**Establish intractability.** If $X \leq_p Y$ and $X$ cannot be solved in polynomial-time, then $Y$ cannot be solved in polynomial time.

**Establish equivalence.** If $X \leq_p Y$ and $Y \leq_p X$, we use notation $X \equiv_p Y$. \\
\text{\textup{up to cost of reduction}}
Reduction By Simple Equivalence

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.
INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in $S$?

Ex. Is there an independent set of size $\geq 6$? Yes.
Ex. Is there an independent set of size $\geq 7$? No.
**Vertex Cover**

**VERTEX COVER:** Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in $S$?

**Ex.** Is there a vertex cover of size $\leq 4$? Yes.

**Ex.** Is there a vertex cover of size $\leq 3$? No.
Claim. \( \text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET} \).

Pf. We show \( S \) is an independent set iff \( V - S \) is a vertex cover.
Vertex Cover and Independent Set

**Claim.** $\text{VERTEX-COVER} \equiv_p \text{INDEPENDENT-SET}$.  

**Pf.** We show $S$ is an independent set iff $V - S$ is a vertex cover.

$\implies$

- Let $S$ be any independent set.
- Consider an arbitrary edge $(u, v)$.
- $S$ independent $\implies u \notin S$ or $v \notin S$ $\implies u \in V - S$ or $v \in V - S$.
- Thus, $V - S$ covers $(u, v)$.

$\impliedby$

- Let $V - S$ be any vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- Observe that $(u, v) \notin E$ since $V - S$ is a vertex cover.
- Thus, no two nodes in $S$ are joined by an edge $\implies S$ independent set.
Reduction from Special Case to General Case

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.
Set Cover

**SET COVER:** Given a set $U$ of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$, does there exist a collection of $\leq k$ of these sets whose union is equal to $U$?

**Sample application.**
- $m$ available pieces of software.
- Set $U$ of $n$ capabilities that we would like our system to have.
- The $i$th piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all $n$ capabilities using fewest pieces of software.

**Ex:**

$U = \{1, 2, 3, 4, 5, 6, 7\}$  
$k = 2$  
$S_1 = \{3, 7\} \quad S_4 = \{2, 4\}$  
$S_2 = \{3, 4, 5, 6\} \quad S_5 = \{5\}$  
$S_3 = \{1\} \quad S_6 = \{1, 2, 6, 7\}$
Claim. \( \text{VERTEX-COVER} \leq_p \text{SET-COVER} \).

Pf. Given a \( \text{VERTEX-COVER} \) instance \( G = (V, E), k \), we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.
- Create \( \text{SET-COVER} \) instance:
  - \( k = k \), \( U = E \), \( S_v = \{ e \in E : e \text{ incident to } v \} \)
- Set-cover of size \( \leq k \) iff vertex cover of size \( \leq k \).
Polynomial-Time Reduction

Basic strategies.
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.
8.2 Reductions via "Gadgets"

Basic reduction strategies.
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."
Satisfiability

**Literal:** A Boolean variable or its negation.  \( x_i \) or \( \overline{x_i} \)

**Clause:** A disjunction of literals. \( C_j = x_1 \lor \overline{x_2} \lor x_3 \)

**Conjunctive normal form:** A propositional formula \( \Phi \) that is the conjunction of clauses. \( \Phi = C_1 \land C_2 \land C_3 \land C_4 \)

**SAT:** Given CNF formula \( \Phi \), does it have a satisfying truth assignment?

**3-SAT:** SAT where each clause contains (at most) 3 literals. each corresponds to a different variable

Ex: \( (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (x_2 \lor x_3) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_3}) \)

**Yes:** \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false} \).
3 Satisfiability Reduces to Independent Set

Claim. $3$-SAT $\leq_p$ INDEPENDENT-SET.

Pf. Given an instance $\Phi$ of $3$-SAT, we construct an instance $(G, k)$ of INDEPENDENT-SET that has an independent set of size $k$ iff $\Phi$ is satisfiable.

Construction.
- $G$ contains $3$ vertices for each clause, one for each literal.
- Connect $3$ literals in a clause in a triangle.
- Connect literal to each of its negations.

$$G$$

\[
\begin{array}{c}
\overline{x}_1 \\
x_2 \\
x_3 \\
\overline{x}_2 \\
x_1 \\
x_3 \\
\overline{x}_1 \\
x_2 \\
x_4
\end{array}
\]

$$k = 3$$

$$\Phi = (\overline{x}_1 \lor x_2 \lor x_3) \land (x_1 \lor \overline{x}_2 \lor x_3) \land (\overline{x}_1 \lor x_2 \lor x_4)$$
3 Satisfiability Reduces to Independent Set

Claim. $G$ contains independent set of size $k = |\Phi|$ iff $\Phi$ is satisfiable.

Pf. $\Rightarrow$ Let $S$ be independent set of size $k$.
- $S$ must contain exactly one vertex in each triangle.
- Set these literals to true. and any other variables in a consistent way
- Truth assignment is consistent and all clauses are satisfied.

Pf $\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size $k$. •

$G$

$k = 3$

$$\Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor x_4)$$
Basic reduction strategies.

- Simple equivalence: $\text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER}$.
- Special case to general case: $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$.
- Encoding with gadgets: $\text{3-SAT} \leq_p \text{INDEPENDENT-SET}$.

Transitivity. If $X \leq_p Y$ and $Y \leq_p Z$, then $X \leq_p Z$.

Pf idea. Compose the two algorithms.

Ex: $\text{3-SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}$. 
**Self-Reducibility**

**Decision problem.** Does there exist a vertex cover of size \( \leq k \)?

**Search problem.** Find vertex cover of minimum cardinality.

**Self-reducibility.** Search problem \( \leq_p \) decision version.
- Applies to all (NP-complete) problems in this chapter.
- Justifies our focus on decision problems.

**Ex: to find min cardinality vertex cover.**
- (Binary) search for cardinality \( k^* \) of min vertex cover.
- Find a vertex \( v \) such that \( G - \{ v \} \) has a vertex cover of size \( \leq k^* - 1 \).
  - any vertex in any min vertex cover will have this property
  - include \( v \) in the vertex cover.
- Recursively find a min vertex cover in \( G - \{ v \} \).
8.3 Definition of NP
Decision Problems

Decision problem.
- $X$ is a set of strings.
- Instance: string $s$.
- Algorithm $A$ solves problem $X$: $A(s) = \text{yes}$ iff $s \in X$.

Polynomial time. Algorithm $A$ runs in poly-time if for every string $s$, $A(s)$ terminates in at most $p(|s|)$ "steps", where $p(\cdot)$ is some polynomial.

PRIMES: $X = \{2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots\}$
**Definition of P**

**P.** Decision problems for which there is a poly-time algorithm.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Algorithm</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLE</td>
<td>Is x a multiple of y?</td>
<td>Grade school division</td>
<td>51, 17</td>
<td>51, 16</td>
</tr>
<tr>
<td>RELPRIME</td>
<td>Are x and y relatively prime?</td>
<td>Euclid (300 BCE)</td>
<td>34, 39</td>
<td>34, 51</td>
</tr>
<tr>
<td>PRIMES</td>
<td>Is x prime?</td>
<td>AKS (2002)</td>
<td>53</td>
<td>51</td>
</tr>
<tr>
<td>EDIT-DISTANCE</td>
<td>Is the edit distance between x and y less than 5?</td>
<td>Dynamic programming</td>
<td>neither</td>
<td>acgggt ttttta</td>
</tr>
<tr>
<td>LSOLVE</td>
<td>Is there a vector x that satisfies Ax = b?</td>
<td>Gauss-Edmonds elimination</td>
<td><img src="matrix.png" alt="Matrix" /></td>
<td><img src="matrix.png" alt="Matrix" /></td>
</tr>
</tbody>
</table>
**Certification algorithm intuition.**

- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether \( s \in X \) on its own; rather, it checks a proposed proof \( t \) that \( s \in X \).

**Def.** Algorithm \( C(s, t) \) is a **certifier** for problem \( X \) if for every string \( s, s \in X \) iff there exists a string \( t \) such that \( C(s, t) = \text{yes} \).

**NP.** Decision problems for which there exists a **poly-time** certifier.

\( C(s, t) \) is a poly-time algorithm and \( |t| \leq p(|s|) \) for some polynomial \( p(\cdot) \).

**Remark.** NP stands for **nondeterministic** polynomial-time.
**Certifiers and Certificates: Composite**

**COMPOSITES.** Given an integer $s$, is $s$ composite?

**Certificate.** A nontrivial factor $t$ of $s$. Note that such a certificate exists iff $s$ is composite. Moreover $|t| \leq |s|$.

**Certifier.**

```java
boolean C(s, t) {
    if (t <= 1 or t >= s)
        return false
    else if (s is a multiple of t)
        return true
    else
        return false
}
```

**Instance.** $s = 437,669$.

**Certificate.** $t = 541$ or $809$. 437,669 = $541 \times 809$

**Conclusion.** *COMPOSITES* is in NP.
Certifiers and Certificates: 3-Satisfiability

**SAT.** Given a CNF formula $\Phi$, is there a satisfying assignment?

**Certificate.** An assignment of truth values to the $n$ boolean variables.

**Certifier.** Check that each clause in $\Phi$ has at least one true literal.

**Ex.**

$$
(\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (x_1 \lor x_2 \lor x_4) \land (\overline{x_1} \lor \overline{x_3} \lor \overline{x_4})
$$

instance $s$

$x_1 = 1, \ x_2 = 1, \ x_3 = 0, \ x_4 = 1$

certificate $t$

**Conclusion.** $\text{SAT}$ is in $\text{NP}$. 
Certifiers and Certificates: Hamiltonian Cycle

**HAM-CYCLE.** Given an undirected graph $G = (V, E)$, does there exist a simple cycle $C$ that visits every node?

**Certificate.** A permutation of the $n$ nodes.

**Certifier.** Check that the permutation contains each node in $V$ exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

**Conclusion.** **HAM-CYCLE** is in NP.
P, NP, EXP

P. Decision problems for which there is a poly-time algorithm.

EXP. Decision problems for which there is an exponential-time algorithm.

NP. Decision problems for which there is a poly-time certifier.

Claim. $P \subseteq NP$.

Pf. Consider any problem $X$ in $P$.

- By definition, there exists a poly-time algorithm $A(s)$ that solves $X$.
- Certificate: $t = \varepsilon$, certifier $C(s, t) = A(s)$. 

Claim. $NP \subseteq EXP$.

Pf. Consider any problem $X$ in $NP$.

- By definition, there exists a poly-time certifier $C(s, t)$ for $X$.
- To solve input $s$, run $C(s, t)$ on all strings $t$ with $|t| \leq p(|s|)$.
- Return $\text{yes}$, if $C(s, t)$ returns $\text{yes}$ for any of these. 

The Main Question: P Versus NP

Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
- Is the decision problem as easy as the certification problem?
- Clay $1 million prize.

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...
If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...

Consensus opinion on P = NP? Probably no.

If P ≠ NP

If P = NP would break RSA cryptography (and potentially collapse economy)
The Simpson's: P = NP?
Futurama: P = NP?

P = NP ?
Looking for a Job?

Some writers for the Simpsons and Futurama.

2-Player Zero-Sum Games

Example: Shooter-Goalie

<table>
<thead>
<tr>
<th></th>
<th>Block Left</th>
<th>Block Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shoot Left</td>
<td>1/2</td>
<td>0.9</td>
</tr>
<tr>
<td>Shoot Right</td>
<td>0.8</td>
<td>1/3</td>
</tr>
</tbody>
</table>

Shooter Minimax (solve in Mathematica)

Maximize \( \left\{ \begin{array}{l}
L_s \geq 0 \land R_s \geq 0 \land L_s + R_s = 1 \\
L_s \left( \frac{1}{2} \right) + R_s \left( \frac{4}{5} \right) \geq v \\
L_s \left( \frac{9}{10} \right) + R_s \left( \frac{1}{3} \right) \geq v
\end{array} \right\} \), \( \{v, L_s, \} \)

Goalie Blocks Left

Goalie Blocks Right
## 2-Player Zero-Sum Games

**Example:** Shooter-Goalie

<table>
<thead>
<tr>
<th></th>
<th>Block Left</th>
<th>Block Right</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shoot Left</strong></td>
<td>1/2</td>
<td>0.9</td>
</tr>
<tr>
<td><strong>shoot Right</strong></td>
<td>0.8</td>
<td>1/3</td>
</tr>
</tbody>
</table>

\[
\text{Maximize}\left\{\begin{aligned}
L & \geq 0, R \geq 0, 1 \leq L + R \leq 1, \\
L(1/2) + R(4/5) & \geq v, \\
L(9/10) + R(1/3) & \geq v
\end{aligned}\right\}, \{v, L, R\}
\]

**Solution:**
\{v\to 0.638462, L\to 0.538462, R\to 0.461538\}

Shooter guaranteed to score at least 63.846\% of the time by shooting left 53.846\% of time.
2-Player Zero-Sum Games

**Example:** Shooter-Goalie

<table>
<thead>
<tr>
<th></th>
<th>Block Left</th>
<th>Block Right</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shoot Left</strong></td>
<td>1/2</td>
<td>0.9</td>
</tr>
<tr>
<td><strong>Shoot Right</strong></td>
<td>0.8</td>
<td>1/3</td>
</tr>
</tbody>
</table>

**Shooter Minimax (solve in Mathematica)**

Maximize

\[
\left\{ \begin{array}{l}
L_G \geq 0 \quad \& \quad R_G \geq 0 \\
L_G \left( \frac{-1}{2} \right) + R_G \left( \frac{-9}{10} \right) \geq v \\
L_G \left( \frac{-4}{5} \right) + R_G \left( \frac{-1}{3} \right) \geq v
\end{array} \right\}, \{v, L_G, R_G\}
\]

- Shooter Shoots Left
- Shooter Shoots Right
2-Player Zero-Sum Games

Example: Shooter-Goalie

<table>
<thead>
<tr>
<th>Shoot Left</th>
<th>Shoot Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Left</td>
<td>1/2</td>
</tr>
<tr>
<td>Block Right</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Maximize
$$\left\{ \begin{array}{l}
L_G \geq 0 \land R_G \geq 0 \land L_G + R_G \geq 1 \\
L_G \left( \frac{-1}{2} \right) + R_G \left( \frac{-9}{10} \right) \geq v \land L_G \left( \frac{-4}{5} \right) + R_G \left( \frac{-1}{3} \right) \geq v
\end{array} \right\} \{v, L_G, R_G\}$$

Solution:
$$\{v \rightarrow -0.638462, L_G \rightarrow 0.653846, R_G \rightarrow 0.346154\}$$

Goalie can ensure goal is scored at most 63.846% of the time by blocking left 65.384% of the time.