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)
- 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>Shooter</th>
<th>Block Left</th>
<th>1/2</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Right</td>
<td>0.8</td>
<td>1/3</td>
<td></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?

Finding Minimax Optimal Solution using Linear Programming

Variables: $p_1, p_2, v$ ($p_i$ is probability of action $i$)
Goal: Maximize $v$ (our expected reward).

Constraints:
- $p_1, p_2 \geq 0$
- $1 \geq p_1 + p_2 \geq 1$
- For all columns $j$ we have
  \[ \sum m_{ij} \geq v \]
  \[ m_{ij} \text{ denotes reward when player 1 takes action } i \text{ and player 2 takes action } j. \]

Duality: $\bar{v}^T = -\bar{v}^T$

Example: Shooter-Goalie

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

Maximize
\[ \begin{bmatrix} S_0 \geq 0, S_1 \geq 0, 0.1 \leq S_2 + S_3 \leq 1, \frac{S_4}{5} + S_5 \end{bmatrix} \]

Subject to:
- $S_0 \geq 0, S_1 \geq 0, 0.1 \leq S_2 + S_3 \leq 1, \frac{S_4}{5} + S_5 \geq \bar{v} (v, S_2, S_3)$
- Goalie Blocks Left
- Goalie Blocks Right

Goalie Blocks Left
Goalie Blocks Right
Example: Shooter-Goalie

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

Maximize: \( v, S^L, S^R \)
\[
\begin{align*}
S^L &\geq 0, S^R \geq 0, 0.1 \leq S^L + S^R \leq 1, \\
S^L(1/2) + S^R(4/5) &\geq v, \\
S^L(3/10) + S^R(1/3) &\geq v
\end{align*}
\]

Solution: \( v \approx 0.638462, S^L \approx 0.538462, S^R \approx 0.461538 \)
- Shooter guaranteed to score at least 63.846% of the time by shooting left 53.846% of the time

Chapter 8
NP and Computational Intractability

Algorithms Design Patterns and Anti-Patterns

Algorithm design patterns:
- Greedy
- Divide-and-conquer
- Dynamic programming
- Duality
- Reductions
- Local search
- Randomization

Algorithm design anti-patterns:
- NP-completeness
- PSPACE-completeness
- Undecidability

NP-completeness: \( \text{O}(\text{polynomial}) \) algorithm unlikely
- PSPACE-completeness: \( \text{O}(\text{polynomial}) \) 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></th>
<th>Yes</th>
<th>Probably no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Largest path</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matching</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3D-matching</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min cut</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max cut</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-SAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-SAT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planar 4-color</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planar 3-color</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bipartite vertex cover</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertex cover</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primality testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Factoring</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Polynomial-Time Reduction

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. 

Independent Set

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 $|S| = k$, 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.

---

Vertex Cover and Independent Set

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

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

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

- **$\Leftarrow$** 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 $\Rightarrow S$ 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.
- $U$ 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:**
- $U = \{1, 2, 3, 4, 5, 6, 7\}$
- $k = 2$
- $S_1 = \{3, 7\}$, $S_2 = \{2, 4\}$, $S_3 = \{3, 4, 5, 6\}$, $S_4 = \{5\}$, $S_5 = \{1, 2, 6, 7\}$
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. $C_j = x_i \lor x_j \lor x_k$

Conjunctive normal form: A propositional formula $\Phi$ that is the conjunction of clauses.

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

3-SAT: SAT where each clause contains at most 3 literals.

$\Phi = (C_1 \land C_2 \land C_3 \land C_4)$

Yes: $x_i$ true, $x_j$ true, $x_k$ false.

No.

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 $k$ if $\Phi$ 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.

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

$k = 3$

$\Leftrightarrow$ $x_1$ true, $x_2$ true, $x_3$ false.

3 Satisfiability Reduces to Independent Set

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

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

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

Review

Basic reduction strategies:
- Simple equivalence: INDEPENDENT-SET $\leq_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ 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: 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ 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.

1. (Binary) search for cardinality \( k^* \) of min vertex cover.
2. 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
3. Include \( v \) in the vertex cover.
4. 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: \( X = \{2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots \} \)

Algorithm. Agrawal-Kayal-Saxena, 2002

<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</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>24, 39</td>
<td>34, 51</td>
</tr>
<tr>
<td>PRIMES</td>
<td>Is ( x ) prime?</td>
<td>AKS (2002)</td>
<td>63</td>
<td>51</td>
</tr>
<tr>
<td>EDIT-DISTANCE</td>
<td>Is the edit distance between ( x ) and ( y ) less than ( S )?</td>
<td>Dynamic programming</td>
<td>54</td>
<td>48</td>
</tr>
<tr>
<td>LSOLVE</td>
<td>Is there a vector ( x ) that satisfies ( Ax = b )?</td>
<td>Gaussian elimination</td>
<td>53</td>
<td>53</td>
</tr>
</tbody>
</table>

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 if \( s \) is composite. Moreover \(|t| \leq |s|\).

Certifier.

```
bool 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.

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

\( s_1 = 1, s_2 = 1, s_3 = 0, s_4 = 1 \)

certificate t

Conclusion. SAT is in 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: \( t = \epsilon \), 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 \( yes \) if \( C(s, t) \) returns \( yes \) 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.

The Simpson’s: P = NP?

Copyright © 1990, Matt Groening

Futurama: P = NP?

Copyright © 2000, Twentieth Century Fox
Looking for a Job?

Some writers for the Simpsons and Futurama.


2-Player Zero-Sum Games

Example: Shooter-Goalie

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

Shooter Minimax (solve in Mathematica)

\[
\max \left\{ \begin{array}{l}
L_o \geq 0 \land R_o \geq 0 \land L_o + R_o = 1 \\
L_o \left( \frac{1}{2} \right) + R_o \left( \frac{3}{2} \right) \geq v
\end{array} \right. \}
\]

Goalie: Blocks Left

Goalie: Blocks Right

Solution:

\[
\{ y \rightarrow -0.638462, L_o \rightarrow 0.638462, R_o \rightarrow 0.346154 \}
\]

Goalie can ensure goal is scored at most 34.61% of the time by blocking left 65.39% of the time.