You think you know when you can learn, are more sure when you can write, even more when you can teach, but certain when you can program.
<table>
<thead>
<tr>
<th>CSPs</th>
<th>Combinatorial Optimization</th>
<th>Adversarial Search</th>
</tr>
</thead>
</table>
Constraint Satisfaction Problems
Types of Search Problems

- **Shortest-path (vacuum, tile puzzle, M&C)**
  - given operators and their costs
  - want least-cost path to a goal
  - goal depth/cost unknown

- **Constraint satisfaction (map coloring, n-queens)**
  - any goal is fine
  - fixed depth
  - explicit constraints on partial solutions
**Variable choice:** choose most constrained variable (smallest domain)
- want to keep tree small, failing quickly

**Value choice:** try least constraining value first (fewest removals)
- might as well succeed sooner if possible
<table>
<thead>
<tr>
<th></th>
<th>BT</th>
<th>FC</th>
<th>FC+MCV</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>&gt; 1M</td>
<td>2K</td>
<td>60</td>
</tr>
<tr>
<td>n-Queens</td>
<td>&gt; 40M</td>
<td>&gt; 40M</td>
<td>820K</td>
</tr>
<tr>
<td>Zebra</td>
<td>3.9M</td>
<td>35K</td>
<td>500</td>
</tr>
<tr>
<td>Random 1</td>
<td>420K</td>
<td>26K</td>
<td>2K</td>
</tr>
<tr>
<td>Random 2</td>
<td>940K</td>
<td>77K</td>
<td>15K</td>
</tr>
</tbody>
</table>
Maintaining Arc Consistency

Ensure every value for $x$ has a legal value in all neighbors $y$. If one doesn’t, remove it and ensure consistency of all $y$. 
Maintaining Arc Consistency

Ensure every value for \( x \) has a legal value in all neighbors \( y \). If one doesn’t, remove it and ensure consistency of all \( y \).

\[
\text{while } Q \text{ is not empty} \\
(x, y) \leftarrow \text{pop } Q \\
\text{if } \text{revised}(x, y) \text{ then} \\
\quad \text{if } x’s \text{ domain is now empty, return failure} \\
\quad \text{for every other neighbor } z \text{ of } x \\
\quad \quad \text{push } (z, x) \text{ on } Q
\]

\[
\text{revised}(x, y) \\
\text{revised} \leftarrow \text{false} \\
\text{foreach } v \text{ in } x’s \text{ domain} \\
\quad \text{if no value in domain of } y \text{ is compatible with } v \\
\quad \quad \text{remove } v \text{ from } x’s \text{ domain} \\
\quad \quad \text{revised} \leftarrow \text{true} \\
\text{return } \text{revised}
\]
Other Algorithms for CSPs

- (Conflict-directed) Backjumping
- Dynamic backtracking
- Randomized restarting

Course projects!
what is a course project?

asst 1 going out on Wed
Combinatorial Optimization

CSPs

Combinatorial Optimization

- Types of Problems
- Hill-Climbing

Adversarial Search
Types of Search Problems

- **Shortest-path (M&C, vacuum, tile puzzle)**
  - want least-cost path to a goal
  - goal depth unknown
  - given operators and their costs

- **Constraint satisfaction (map coloring, \( n \)-queens)**
  - any goal is fine
  - maximum depth = number of variables
  - given explicit constraints on variables

- **Combinatorial optimization (TSP, max-CSP)**
  - want least-cost goal
  - maximum depth = number of variables
  - every leaf is a solution
Sol ← some random solution (probably poor quality).

Do limit times

New ← random neighbor of Sol.

If New better than Sol,
then Sol ← New.
Hill-Climbing

Sol ← some random solution (probably poor quality).
Do limit times
New ← random neighbor of Sol.
If New better than Sol,
then Sol ← New.

Elaborations: best neighbor (aka gradient-descent)
restarts
simulated annealing
population (GAs, ‘go with the winners’)
Adversarial Search
Another Twist on Search

- Shortest-path (M&C, vacuum, tile puzzle)
  - want least-cost path to goal at unknown depth
- Constraint satisfaction (map coloring, $n$-queens)
  - any goal that satisfies constraints (fixed depth)
- Combinatorial optimization (TSP, max-CSP)
  - want least-cost goal (fixed depth)
- Decisions with an adversary (chess, tic-tac-toe)
  - adversary might prevent path to best goal
  - want best assured outcome
Adversarial Search: Minimax

Each *ply* corresponds to half a *move*. Terminal states are labeled with value. Can also bound depth and use a *static evaluation function* on non-terminal states.
A *3-length* is a complete row, column, or diagonal.

value of position  =  ∞ if win for me,

or  =  −∞ if a win for you,

otherwise  =  # 3-lengths open for me − # 3-lengths open for you
Tic-tac-toe: two-ply search

CSPs
Combinatorial Optimization
Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- EOLQs

Fig. 3.8 Minimax applied to tic-tac-toe (stage 1).
Tic-tac-toe: second move

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- EOLQs

Fig. 3.9 Minimax applied to tic-tac-toe (stage 2).
CSPs
Combinatorial Optimization

Adversarial Search
- Another Type
- Minimax
- Tic-tac-toe
- Improvements
- EOLQs

Tic-tac-toe: third move

Fig. 3.10 Minimax applied to tic-tac-toe (stage 3).
## Improving the Search

<table>
<thead>
<tr>
<th>CSPs</th>
<th>Combinatorial Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Adversarial Search</td>
</tr>
<tr>
<td></td>
<td>Another Type</td>
</tr>
<tr>
<td></td>
<td>Minimax</td>
</tr>
<tr>
<td>** Tic-tac-toe **</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Improvements</td>
</tr>
<tr>
<td></td>
<td>EOLQs</td>
</tr>
</tbody>
</table>

- partial expansion, SEF
- symmetry (‘transposition tables’)
- search more ply as we have time (De Groot figure)
- avoid unnecessary evaluations
Please write down the most pressing question you have about the course material covered so far and put it in the box on your way out.

Thanks!