Adversarial Search and Game-Playing
Environment Type Discussed In this Lecture

- Turn-taking: Semi-dynamic
- Deterministic and non-deterministic
Adversarial Search

- Examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.

- A good example is in board games.

- Adversarial games, while much studied in AI, are a small part of game theory in economics.
Typical AI assumptions

- Two agents whose actions alternate
- Utility values for each agent are the opposite of the other
  - creates the adversarial situation
- Fully observable environments
- In game theory terms: Zero-sum games of perfect information.
- We’ll relax these assumptions later.
Search versus Games

- **Search – no adversary**
  - Solution is (heuristic) method for finding goal
  - Heuristic techniques can find *optimal* solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

- **Games – adversary**
  - Solution is *strategy* (strategy specifies move for every possible opponent reply).
  - *Optimality depends on opponent.* Why?
  - Time limits force an *approximate* solution
  - Evaluation function: evaluate “goodness” of game position
  - Examples: chess, checkers, Othello, backgammon
## Types of Games

<table>
<thead>
<tr>
<th></th>
<th>Deterministic</th>
<th>Chance Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect information</td>
<td>Chess, checkers,</td>
<td>Backgammon,</td>
</tr>
<tr>
<td></td>
<td>go, othello</td>
<td>monopoly</td>
</tr>
<tr>
<td>Imperfect information</td>
<td>Bridge, Skat</td>
<td>Poker, scrabble,</td>
</tr>
<tr>
<td>(Initial Chance</td>
<td></td>
<td>blackjack</td>
</tr>
<tr>
<td>Moves)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• **Theorem of Nobel Laureate Harsanyi**: Every game with chance moves during the game has an equivalent representation with initial chance moves only.
• A deep result, but computationally it is more tractable to consider chance moves as the game goes along.
• This is basically the same as the issue of full observability + nondeterminism vs. partial observability + determinism.
Game Setup

- Two players: MAX and MIN

- MAX moves first and they take turns until the game is over
  - Winner gets award, loser gets penalty.

- Games as search:
  - Initial state: e.g. board configuration of chess
  - Successor function: list of (move,state) pairs specifying legal moves.
  - Terminal test: Is the game finished?
  - Utility function: Gives numerical value of terminal states. E.g. win (+1), lose (-1) and draw (0) in tic-tac-toe or chess

- MAX uses search tree to determine next move.
Size of search trees

- $b = \text{branching factor}$
- $d = \text{number of moves by both players}$
- Search tree is $O(b^d)$

- **Chess**
  - $b \approx 35$
  - $D \approx 100$
    - search tree is $\sim 10^{154}$ (!!!)
    - completely impractical to search this

- Game-playing emphasizes being able to make optimal decisions in a finite amount of time
  - Somewhat realistic as a model of a real-world agent
  - Even if games themselves are artificial
Partial Game Tree for Tic-Tac-Toe
Game tree (2-player, deterministic, turns)

How do we search this tree to find the optimal move?
Minimax strategy: Look ahead and reason backwards

- Find the optimal *strategy* for MAX assuming an infallible MIN opponent
  - Need to compute this all the down the tree
  - [Game Tree Search Demo](#)

- Assumption: Both players play optimally!
- Given a game tree, the optimal strategy can be determined by using the *minimax value of each node*.
- Zermelo 1912.
Two-Ply Game Tree
Two-Ply Game Tree
Two-Ply Game Tree
Two-Ply Game Tree

Minimax maximizes the utility for the worst-case outcome for max

The minimax decision
Pseudocode for Minimax Algorithm

**function MINIMAX-DECISION(state) returns an action**

inputs: state, current state in game

\[ v \leftarrow \text{MAX-VALUE(state)} \]

return the action in SUCCESSORS(state) with value \( v \)

**function MAX-VALUE(state) returns a utility value**

if TERMINAL-TEST(state) then return UTILITY(state)

\[ v \leftarrow -\infty \]

for \( a, s \) in SUCCESSORS(state) do

\[ v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s)) \]

return \( v \)

**function MIN-VALUE(state) returns a utility value**

if TERMINAL-TEST(state) then return UTILITY(state)

\[ v \leftarrow \infty \]

for \( a, s \) in SUCCESSORS(state) do

\[ v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s)) \]

return \( v \)
Example of Algorithm Execution

MAX to move

Diagram showing a tree structure with nodes labeled from 0 to 4, each containing a number, and arrows indicating the direction of movement.
### Minimax Algorithm

- **Complete depth-first exploration of the game tree**

- **Assumptions:**
  - Max depth = d, b legal moves at each point
  - E.g., Chess: d \( \sim 100 \), b \( \sim 35 \)

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Minimax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>( \mathcal{O}(b^d) )</td>
</tr>
<tr>
<td>Space</td>
<td>( \mathcal{O}(bd) )</td>
</tr>
</tbody>
</table>
Practical problem with minimax search

- Number of game states is exponential in the number of moves.
  - Solution: Do not examine every node
    - => pruning
      - Remove branches that do not influence final decision

- Revisit example ...
Alpha-Beta Example

Do DF-search until first leaf

Range of possible values

\([-\infty, +\infty]\)
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)

\[
\begin{array}{c}
\text{MAX} \\
\text{MIN}
\end{array}
\]

[3, +\infty]

[3, 3]
This node is worse for MAX
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)
Alpha-beta Algorithm

- Depth first search – only considers nodes along a single path at any time

\[ \alpha = \text{highest-value choice that we can guarantee for MAX so far in the current subtree.} \]

\[ \beta = \text{lowest-value choice that we can guarantee for MIN so far in the current subtree.} \]

- update values of \( \alpha \) and \( \beta \) during search and prunes remaining branches as soon as the value is known to be worse than the current \( \alpha \) or \( \beta \) value for MAX or MIN.

- Alpha-beta Demo.
Effectiveness of Alpha-Beta Search

- **Worst-Case**
  - branches are ordered so that no pruning takes place. In this case alpha-beta gives no improvement over exhaustive search

- **Best-Case**
  - each player’s best move is the left-most alternative (i.e., evaluated first)
  - in practice, performance is closer to best rather than worst-case

- **In practice often get $O(b^{(d/2)})$ rather than $O(b^d)$**
  - this is the same as having a branching factor of $\sqrt{b}$,
    - since $(\sqrt{b})^d = b^{(d/2)}$
    - i.e., we have effectively gone from $b$ to square root of $b$
  - e.g., in chess go from $b \sim 35$ to $b \sim 6$
    - this permits much deeper search in the same amount of time
    - Typically twice as deep.
Example

-which nodes can be pruned?
Final Comments about Alpha-Beta Pruning

- Pruning does not affect final results
- Entire subtrees can be pruned.
- Good move *ordering* improves effectiveness of pruning
- Repeated states are again possible.
  - Store them in memory = transposition table
Practical Implementation

How do we make these ideas practical in real game trees?

Standard approach:

- **cutoff test**: (where do we stop descending the tree)
  - depth limit
  - better: iterative deepening
  - cutoff only when no big changes are expected to occur next (quiescence search).

- **evaluation function**
  - When the search is cut off, we evaluate the current state by estimating its utility using an **evaluation function**.
Static (Heuristic) Evaluation Functions

- **An Evaluation Function:**
  - estimates how good the current board configuration is for a player.
  - Typically, one figures how good it is for the player, and how good it is for the opponent, and subtracts the opponents score from the players.
  - Othello: Number of white pieces - Number of black pieces
  - Chess: Value of all white pieces - Value of all black pieces

- Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
- If the board evaluation is X for a player, it’s -X for the opponent.
- Many clever ideas about how to use the evaluation function.
  - e.g. null move heuristic: let opponent move twice.

- **Example:**
  - Evaluating chess boards,
  - Checkers
  - Tic-tac-toe
For chess, typically \textit{linear} weighted sum of \textit{features}

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

\text{e.g., } w_1 = 9 \text{ with } f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.}
Iterative (Progressive) Deepening

- In real games, there is usually a time limit $T$ on making a move

- How do we take this into account?
  - using alpha-beta we cannot use “partial” results with any confidence unless the full breadth of the tree has been searched
  - So, we could be conservative and set a conservative depth-limit which guarantees that we will find a move in time $< T$
    - disadvantage is that we may finish early, could do more search

- In practice, iterative deepening search (IDS) is used
  - IDS runs depth-first search with an increasing depth-limit
  - when the clock runs out we use the solution found at the previous depth limit
The State of Play

- **Checkers:**

- **Chess:**

- **Othello:**
  - Human champions refuse to compete against computers: they are too good.

- **Go:**
  - Human champions refuse to compete against computers: they are too bad \( b > 300 \) (!)

- See (e.g.) [http://www.cs.ualberta.ca/~games/](http://www.cs.ualberta.ca/~games/) for more information
The University of Alberta GAMES Group

Game-playing,
Analytical methods,
Minimax search and
Empirical
Studies

Projects | News | What We Do | People | Links | Publications

Announcements

- Weekly GAMES group meetings are from 4-5pm on Thursdays at CSC333. You can check the schedule here.
- The University of Alberta GAMES Group has an opening for a postdoctoral fellow in the area of Artificial Intelligence in Commercial (Video) Games. Check here for details.

Projects

- **Checkers**
  - Chinook is the official world checkers champion.
- **Poker**
  - Poki is the strongest poker AI in the world.
- **Lines of Action**
  - YL & Mona are two of the best LoA programs in the world.
- **Hex**
  - Queenbee is one of the best Hex programs in the world.
- **Go**
  - The Computer Go group has developed two programs, Explorer and NeuroGo.
- **Real-Time Strategy**
  - We are trying to apply AI to real-time strategy games.
- **Othello**
  - Logistello defeated the human world Othello champion, 6–0, in 1997. Karnov is another strong program.
- **Amazons**
  - Three programs, and several theoretical contributions.
- **Spades & Hearts**
  - Spades & Hearts are the test beds for the research on multi-player games.
- **Sokoban**
  - Rolling Stone pushes the boundaries of single agent search.
- **Shogi**
  - Seko has won the world computer Shogi championship many times (currently inactive).

Previous Projects:

- **Chess**
- **Aware**
- **Chinese Chess**
- **Post's Correspondence Problem**
- **Dominering**
Deep Blue

1957: Herbert Simon
- “within 10 years a computer will beat the world chess champion”

1997: Deep Blue beats Kasparov
- Parallel machine with 30 processors for “software” and 480 VLSI processors for “hardware search”
- Searched 126 million nodes per second on average
  - Generated up to 30 billion positions per move
  - Reached depth 14 routinely
- Uses iterative-deepening alpha-beta search with transpositioning
  - Can explore beyond depth-limit for interesting moves
Game playing can be effectively modeled as a search problem.

Game trees represent alternate computer/opponent moves.

Evaluation functions estimate the quality of a given board configuration for the Max player.

Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them.

Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper.

For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.
AI Games vs. Economics Game Theory


- Agents can be in cooperation as well as in conflict.

- Agents may move simultaneously/independently.
### Example: The Prisoner’s Dilemma

<table>
<thead>
<tr>
<th>Row Player</th>
<th>Column Player</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>2, 2</td>
</tr>
<tr>
<td>D</td>
<td>3, 0</td>
</tr>
</tbody>
</table>

**Other Famous Matrix Games:**
- Chicken
- Battle of The Sexes
- Coordination
Solving Zero-Sum Games

• Perfect Information: Use Minimax Tree Search.
• Imperfect Information: Extend Minimax Idea with probabilistic actions.
  ➞ von Neumann’s Minimax Theorem: there exists an essentially unique optimal probability distribution for randomizing an agent’s behaviour.
Matching Pennies

- Why should the players randomize?
- What are the best probabilities to use in their actions?

<table>
<thead>
<tr>
<th></th>
<th>Heads</th>
<th>Tails</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heads</td>
<td>1,-1</td>
<td>-1,1</td>
</tr>
<tr>
<td>Tails</td>
<td>-1,1</td>
<td>1,-1</td>
</tr>
</tbody>
</table>
The idea of “look ahead, reason backward” works for any game tree with perfect information. 
- I.e., also in cooperative games.

- In AI, this is called **retrograde analysis**.
- In game theory, it is called **backward induction** or **subgame perfect equilibrium**.

- Can be extended to many games with imperfect information (sequential equilibrium).
Backward Induction Example: Hume’s Farmer Problem

Diagram:

- Node 1: H → Node 2, Not H → Node 2
- Node 2:
  - H1: H1 → Node 2
    - H1: 2, 2
    - notH1: 1, 0
  - H2: H2 → Node 2
    - H2: 3, 0
    - notH2: 1, 1
### Summary: Solving Games

<table>
<thead>
<tr>
<th></th>
<th>Zero-sum</th>
<th>Non zero-sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect Information</td>
<td>Minimax, alpha-beta</td>
<td>Backward induction, retrograde analysis</td>
</tr>
<tr>
<td>Imperfect Information</td>
<td>Probabilistic minimax</td>
<td>Nash equilibrium</td>
</tr>
</tbody>
</table>

**Nash equilibrium** is beyond the scope of this course.
Single Agent vs. 2-Players

- *Every single agent problem can be considered as a special case of a 2-player game.* How?
  1. Make one of the players the Environment, with a constant utility function (e.g., always 0).
    1. The Environment acts but does not care.
  2. An adversarial Environment, with utility function the negative of agent’s utility.
    1. In minimization, Environment’s utility is player’s costs.
    2. Worst-Case Analysis.
    3. E.g., program correctness: no matter what input user gives, program gives correct answer.

- So agent design is a subfield of game theory.
Single Agent Design = Game Theory

Von Neumann-Morgenstern Games

Decision Theory = 2-player game, 1st player the “agent”, 2nd player “environment/nature” (with constant or adversarial utility function)

Markov Decision Processes

Planning Problems

From General To Special Case
Example: And-Or Trees

- If an agent’s actions have nondeterministic effects, we can model worst-case analysis as a zero-sum game where the environment chooses the effects of an agent’s actions.
- Minimax Search ≈ And-Or Search.
- Example: The Erratic Vacuum Cleaner.
  - When applied to dirty square, vacuum cleans it and sometimes adjacent square too.
  - When applied to clean square, sometimes vacuum makes it dirty.
  - Reflex agent: same action for same location, dirt status.
And-Or Tree for the Erratic Vacuum

- The agent “moves” at labelled OR nodes.
- The environment “moves” at unlabelled AND nodes.
- The agent wins if it reaches a goal state.
- The environment “wins” if the agent goes into a loop.
Summary

- Game Theory is a very general, highly developed framework for multi-agent interactions.
- Deep results about equivalences of various environment types.
- See Chapter 17 for more details.