COMP 102: Excursions in Computer Science
Lecture 21: Game Playing

Instructor: Joelle Pineau (jpineau@cs.mcgill.ca)

Class web page: www.cs.mcgill.ca/~jpineau/comp102

Overview of AI

• Typically three components:

THE WORLD

Perception ——> Action ——> Reasoning
Example AI system: Chess playing

IBM Deep Blue defeated Garry Kasparov (1997)
- **Perception:** advanced features of the board.
  - **Actions:** choose a move.
- **Reasoning:** search and evaluation of possible board positions.

Game playing

- **One of the oldest, most well-studied domains in AI!**
  - **Why?**
    - People like them! People are good at playing them!
    - Often viewed as an indicator of intelligence.
      - State spaces are very large and complicated.
      - Sometimes there is stochasticity and imperfect information.
    - Clear, clean description of the environment.
    - Easy performance indicator.

“Games are to AI as Grand Prix racing is to automobile design”. 
Start with an easy game: Tic-Tac-Toe

Defining a search problem for games

- **State space** $S$: all possible configurations of the domain.
  - Initial state $s_0 \in S$: the start state
  - Goal states $G \subset S$: the set of end states

- **Actions** $A$: the set of moves available
Defining a search problem for games

- **Path**: a sequence of states and operators.

- **Solution** of search problem: a path from $s_0$ to $s_g \in G$

- **Utility**: a numerical value associated with a state (higher is better, lower is worse).
  
  E.g.  
  
  +1 if it’s a win,  
  -1 if it’s a loss,  
  0 if it’s a draw or game not terminated.

Representing search: Graphs and Trees

- Visualize the state space search in terms of a graph.

- Graph defined by a set of nodes and a set of edges connecting the vertices.
  
  - Nodes correspond to states.
  - Edges correspond to actions.

- We search for a solution by building a search trees and traversing it to find a goal state.
We want to find a strategy (i.e. way of picking moves) that wins the game.

Game search challenge

- Not quite the same as simple graph searching.
- There is an opponent! The opponent is malicious!
  - Opponent is trying to make things good for itself, and bad for us.
  - We have to simulate the opponent’s decisions.

- **Key idea:**
  - Define a max player (who wants to maximize the utility)
  - And a min player (who wants to minimize the utility.)
Example: Tic-Tac-Toe

Minimax search

- Expand complete search tree, until terminal states have been reached and their utilities computed.

- Go back up from leaves towards the current state of the game.
  - At each min node: backup the worst value among the children.
  - At each max node: backup the best value among the children.
A simple Minimax example

MAX

MIN

MAX

Properties of Minimax search

• Can we use minimax to solve any game?
  – Solve Tic-Tac-Toe? Yes!
  – Solve chess? No.

• Why not?
  – Large number of actions possible (i.e. large branching factor) $b=35$.
  – Path to goal may be very long (i.e. deep tree) $m=100$
  – Large number of states!
Coping with resource limitations

• Suppose we have 100 seconds to make a move, and we can search $10^4$ nodes per second.
  – Can only search $10^6$ nodes!
    (Or even fewer, if we spend time deciding which nodes to search.)

• Possible approach:
  – Only search to a pre-determined depth.
  – Use an evaluation function for the nodes where we cutoff the search.

Cutting the search effort

• Use an evaluation function to evaluate non-terminal nodes.
  – Helps us make a decision without searching until the end of the game.

• Minimax cutoff algorithm:
  Same as standard Minimax, except stop at some maximum depth $m$ and use the evaluation function on those nodes.
Evaluation functions

- An evaluation function $v(s)$ represents the “goodness” of a board state (e.g. chance of winning from that position).
  - Similar to a utility function, but approximate.

- If the features of the board can be evaluated independently, use a linear combination:
  $$v(s) = f_1(s) + f_2(s) + \ldots + f_n(s)$$
  (where $s$ is board state)

- This function can be given by the designer or learned from experience.

Example: Chess

- Evaluation function: $v(s) = f_1(s) + f_2(s)$
  $$f_1(s) = w_1 \ast [(\# \text{ white queens}) - (\# \text{ black queens})]$$
  $$f_2(s) = w_2 \ast [(\# \text{ white pawns}) - (\# \text{ black pawns})]$$
How precise should the evaluation fn be?

- Evaluation function is only approximate, and is usually better if we are close to the end of the game.
- Only the order of the numbers matter: payoffs in deterministic games act as an **ordinal utility function**.

![Game tree diagram]

Minimax cutoff in Chess

- How many moves ahead can we search in Chess?
  >> $10^6$ nodes with $b=35$ allows us to search 4 moves ahead!

- Is this useful?
  - 4-moves ahead ≈ novice player
  - 8-moves ahead ≈ human master, typical PC
  - 12-moves ahead ≈ Deep Blue, Kasparov

- Key idea:
  Search few lines of play, but search them deeply. **Need pruning!**
\(\alpha\)-\(\beta\) Pruning example

\[
\begin{array}{c}
\geq 3 \\
3 \\
3 \quad 12 \quad 8
\end{array}
\]
\(\alpha-\beta\) Pruning example

\[
\begin{array}{c}
\geq 3 \\
3 \\
\leq 2 \\
X \\
X \\
\end{array}
\]

\[
\begin{array}{cccc}
3 & 12 & 8 & 2 \\
\leq 2 & \geq 3 & \leq 14 \\
X & X \\
\end{array}
\]
\(\alpha-\beta\) Pruning example

\[
\begin{array}{c}
\geq 3 \\
\geq 2 \\
\leq 5
\end{array}
\]

\[
\begin{array}{cccc}
3 & 12 & 8 & 2 \\
3 & 14 & 5 & 2
\end{array}
\]

\(\alpha-\beta\) Pruning example

\[
\begin{array}{c}
\times 3 \\
\geq 2 \\
\leq 2
\end{array}
\]

\[
\begin{array}{cccc}
3 & 12 & 8 & 2 \\
3 & 14 & 5 & 2
\end{array}
\]
\(\alpha-\beta\) Pruning

- **Basic idea**: if a path looks worse than what we already have, ignore it.
  - If the best move at a node cannot change (regardless of what we would find by searching) then no need to search further!

- Algorithm is like Minimax, but keeps track of best leaf value for our player (\(\alpha\)) and best one for the opponent (\(\beta\))

Properties of \(\alpha-\beta\) pruning

- Pruning does not affect the final result! You will not play worse than without it.

- **Good move ordering is key** to the effectiveness of pruning.
  - With perfect ordering, explore approximately \(b^{m/2}\) nodes.
    - Means double the search depth, for same resources.
    - In chess: this is difference between novice and expert player.
  - With bad move ordering, explore approximately \(b^m\) nodes.
    - Means nothing was pruned.
  - Evaluation function can be used to order the nodes.

The \(\alpha-\beta\) pruning demonstrates the value of reasoning about which computations are important!
Human or computer - who is better?

Checkers:
- 1994: Chinook (U.of A.) beat world champion Marion Tinsley, ending 40-yr reign.

Othello:
- 1997: Logistello (NEC research) beat the human world champion.
- Today: world champions refuse to play AI computer program (because it’s too good).

Chess:
- 1997: Deep Blue (IBM) beat world champion Gary Kasparov

Backgammon:
- TD-Gammon (IBM) is world champion amongst humans and computers

Go:
- Human champions refuse to play top AI player (because it’s too weak)

Bridge:
- Still out of reach for AI players because of coordination issue.

Jeopardy!

- In Winter 2011, Watson, a computer program created by IBM, made history by winning at Jeopardy!
  - Main innovation of Watson: ability to answer questions posed in natural language.
Jeopardy!

• How it works:

  – Watson isn’t connected to the internet, but had access to 4TB of stored information (incl. all of Wikipedia).

  – When given a question, it extracts keywords, looks in database for related facts, compiles list of answers, and ranks them by confidence.

  – Watson is much better at buzzing in than its human opponents. So as long as it knows the answer, it has an edge.

Take-home message

• Understand the basic components (state space, start state, end state, utility function, etc.) required to represent the types of games discussed today.

• Know how to build the search tree.

• Understand the how and why of Minimax, Alpha-beta pruning, and evaluation functions.

• Have some intuition for what makes certain games harder than others.