Adversarial search

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Announcements

- **Homework assignment 4 is out**
  - Programming and experiments
  - Simulated annealing + Genetic algorithm
  - Competition

**Course web page:**
http://www.cs.pitt.edu/~milos/courses/cs1571/
Search review

Search
• Path search
• Configuration search

Optimality
• Finding a path versus finding the optimal path
• Finding a configuration satisfying constraints versus finding the best configuration

Game search
• Game-playing programs developed by AI researchers since the beginning of the modern AI era
  – Programs playing chess, checkers, etc (1950s)

• Specifics of the game search:
  – Sequences of player’s decisions we control
  – Decisions of other player(s) we do not control

• Contingency problem: many possible opponent’s moves must be “covered” by the solution
  Opponent’s behavior introduces an uncertainty in to the game
  – We do not know exactly what the response is going to be
• Rational opponent – maximizes it own utility (payoff) function
Types of game problems

• Types of game problems:
  – Adversarial games:
    • win of one player is a loss of the other
  – Cooperative games:
    • players have common interests and utility function
  – A spectrum of game problems in between the two:

Adversarial games

[Diagram]

Fully cooperative games

we focus on adversarial games only!!

Example of an adversarial 2 person game: Tic-tac-toe

• Player 1 (x) moves first

[Diagram]

Loss Draw Win
Game search problem

- Game problem formulation:
  - Initial state: initial board position + info whose move it is
  - Operators: legal moves a player can make
  - Goal (terminal test): determines when the game is over
  - Utility (payoff) function: measures the outcome of the game and its desirability

- Search objective:
  - find the sequence of player’s decisions (moves) maximizing its utility (payoff)
  - Consider the opponent’s moves and their utility

Game problem formulation (Tic-tac-toe)

Objectives:
- Player 1: maximize outcome
- Player 2: minimize outcome

Initial state

Operators

Terminal (goal) states

Utility: -1 0 1
Minimax algorithm

How to deal with the contingency problem?

- Assuming that the opponent is rational and always optimizes its behavior (opposite to us) we consider the best opponent’s response
- Then the minimax algorithm determines the best move
Minimax algorithm. Example
Minimax algorithm. Example
Minimax algorithm. Example
Minimax algorithm. Example

MAX

MIN

MAX

Example:

\[
\begin{array}{cccccccccc}
4 & 3 & 6 & 2 & 2 & 1 & 9 & 5 & 3 & 1 \\
4 & 6 & 2 & 2 & 1 & 9 & 5 & 3 & 1 & 5
\end{array}
\]

Minimax algorithm. Example

MAX

MIN

MAX

Example:

\[
\begin{array}{cccccccccc}
4 & 3 & 6 & 2 & 2 & 1 & 9 & 5 & 3 & 1 \\
4 & 6 & 2 & 2 & 1 & 9 & 5 & 3 & 1 & 5
\end{array}
\]

Minimax algorithm. Example

MAX

MIN

MAX

Example:

\[
\begin{array}{cccccccccc}
4 & 3 & 6 & 2 & 2 & 1 & 9 & 5 & 3 & 1 \\
4 & 6 & 2 & 2 & 1 & 9 & 5 & 3 & 1 & 5
\end{array}
\]
Minimax algorithm

function MINIMAX-DECISION(game) returns an operator
    for each $op$ in OPERATORS[game] do
        $VALUE[op] = \text{MINIMAX-VALUE}\left(\text{APPLY}(op, game), game\right)$
    end
    return the $op$ with the highest $VALUE[op]$

function MINIMAX-VALUE(state, game) returns a utility value
    if TERMINAL-TEST[game](state) then
        return UTILITY[game](state)
    else if $\max$ is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)

Complexity of the minimax algorithm

- We need to explore the complete game tree before making the decision

Complexity: $m^b$
Complexity of the minimax algorithm

• We need to explore the complete game tree before making the decision

Complexity:

\[ O(b^m) \]

• Impossible for large games
  – Chess: 35 operators, game can have 50 or more moves

Solution to the complexity problem

Two solutions:

1. Dynamic pruning of redundant branches of the search tree
   – identify a provably suboptimal branch of the search tree before it is fully explored
   – Eliminate the suboptimal branch
   
   Procedure: Alpha-Beta pruning

2. Early cutoff of the search tree
   – uses imperfect minimax value estimate of non-terminal states (positions)
Alpha beta pruning

- Some branches will never be played by rational players since they include sub-optimal decisions (for either player)
Alpha beta pruning. Example

MAX

MIN

MAX

\[ \geq 4 \]

\[ \leq 4 \]

\[ 4 \ 3 \ 6 \ 2 \ 2 \ 1 \ 9 \ 5 \ 3 \ 1 \ 5 \ 4 \ 7 \ 5 \]
Alpha beta pruning. Example
Alpha beta pruning. Example

MAX

MIN

= 4

= 4

≥ 6

MAX

= 4

≥ 6

= 2

= 2

≤ 2

= 4

≥ 4

≤ 2

≥ 5

4 3 6 2 2 1 9 5 3 1 5 4 7 5
Alpha beta pruning. Example

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5

4 = 6 ≥ 4 = 4 ≥ 2 = 5 ≤ 5 ≤ 7 ≥ 7
Alpha beta pruning. Example

Nodes that were never explored !!!!
Alpha-Beta pruning

function \texttt{MAX-VALUE}(state, game, α, β) returns the minimax value of \texttt{state}
inputs state, current state in game
game, game description
α, the best score for \texttt{MAX} along the path to state
β, the best score for \texttt{MIN} along the path to state

\textbf{if} \texttt{GOAL-Test(state)} \textbf{then return} \texttt{EVAL(state)}
\textbf{for each} s in \texttt{SUCCESSORS(state)} \textbf{do}
\hspace{1em} α ← \texttt{MAX}(α, \texttt{MIN-VALUE}(s, game, α, β))
\hspace{1em} \textbf{if} α ≥ β \textbf{then return} β
\textbf{end}
\textbf{return} α

function \texttt{MIN-VALUE}(state, game, α, β) returns the minimax value of \texttt{state}
\textbf{if} \texttt{GOAL-Test(state)} \textbf{then return} \texttt{EVAL(state)}
\textbf{for each} s in \texttt{SUCCESSORS(state)} \textbf{do}
\hspace{1em} β ← \texttt{MIN}(β, \texttt{MAX-VALUE}(s, game, α, β))
\hspace{1em} \textbf{if} β ≤ α \textbf{then return} α
\textbf{end}
\textbf{return} β

Using minimax value estimates

\textbf{• Idea:}
\hspace{1em} – Cutoff the search tree before the terminal state is reached
\hspace{1em} – Use imperfect estimate of the minimax value at the leaves
\hspace{1em} \hspace{1em} • Evaluation function

\begin{itemize}
  \item MAX
  \item MIN
  \item Heuristic evaluation function
  \item Cutoff level
\end{itemize}
Design of evaluation functions

- **Heuristic estimate** of the value for a sub-tree
- **Examples of a heuristic function**:
  - **Material advantage in chess, checkers**
    - Gives a value to every piece on the board, its position and combines them
  - More general **feature-based evaluation function**
    - Typically a linear evaluation function:
      \[
      f(s) = f_1(s)w_1 + f_2(s)w_2 + \ldots + f_k(s)w_k
      \]
      \[
      f_i(s) \quad - \text{a feature of a state } s
      \]
      \[
      w_i \quad - \text{feature weight}
      \]

Further extensions to real games

- Restricted set of moves to be considered under **the cutoff level** to reduce branching and improve the evaluation function
  - E.g., consider only the capture moves in chess

...