Parametric optimization

Adversarial search

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

Parametric optimization

**Optimal configuration search:**
- Configurations are described in terms of variables and their values
- Each configuration has a quality measure
- Goal: find the configuration with the best value

When the state space we search is finite, the search problem is called a **combinatorial optimization problem**

When parameters we want to find are real-valued
- **parametric optimization problem**
Parametric optimization

Parametric optimization:

• Configurations are described by a vector of free parameters (variables) $w$ with real-valued values

• **Goal:** find the set of parameters $w$ that optimize the quality measure $f(w)$

Parametric optimization techniques

• Special cases (with efficient solutions):
  – Linear programming
  – Quadratic programming

• First-order methods:
  – Gradient-ascent (descent)
  – Conjugate gradient

• Second-order methods:
  – Newton-Rhapson methods
  – Levenberg-Marquardt

• Constrained optimization:
  – Lagrange multipliers
Linear programming

• **A special case and when:**
  – The objective function is a linear combination of variable values
  – Values the variables can take are constrained by a set of linear constraints

• **Assume variables:** \( x_1, x_2, ..., x_k \)

\[
\text{Minimize } f(x_1, x_2, ..., x_k) = a_1 x_1 + a_2 x_2 + ... + a_k x_k
\]

Subject to constraints:
\[
\begin{align*}
b_{1,1} x_1 + b_{1,2} x_2 + ... + b_{1,k} x_k & \leq 0 \\
b_{2,1} x_1 + b_{2,2} x_2 + ... + b_{2,k} x_k & \leq 0 \\
... \\
b_{m,1} x_1 + b_{m,2} x_2 + ... + b_{m,k} x_k & \leq 0
\end{align*}
\]

Gradient ascent method

• **Gradient ascent:** the same as hill-climbing, but in the continuous parametric space \( w \)

\[
f(w) \quad \frac{\partial}{\partial w} f(w) \big|_{w^*}
\]

• What is the derivative of an increasing function?
Gradient ascent method

- **Gradient ascent**: the same as hill-climbing, but in the continuous parametric space $w$

\[ f(w) \]

\[ \frac{\partial}{\partial w} f(w) \bigg|_{w^*} \]

- What is the derivative of an increasing function?  
  - positive

\[ w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w) \bigg|_{w^*} \]
Gradient ascent method

- New value of the parameter
  \[ w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w) \big|_{w^*} \]
  \[ \alpha > 0 \quad \text{a learning rate (scales the gradient changes)} \]

- Problems: local optima, saddle points, slow convergence
- More complex optimization techniques use additional information (e.g. second derivatives)
Adversarial search

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 into the game
    - We do not know exactly what the response is going to be

- **Rational opponent** – maximizes its 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:**

<table>
<thead>
<tr>
<th>Adversarial games</th>
<th>Fully cooperative games</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

we focus on adversarial games only!!
Example of an adversarial 2 person game: Tic-tac-toe

- Player 1 (x) moves first

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

Operators

Initial state

Terminal (goal) states

Utility: 1 -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

MAX

MIN

MAX

4 3 6 2 2 1 9 5 3 1 5 4 7 5
Minimax algorithm. Example
Minimax algorithm. Example

MAX

MIN

MAX

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

MIN

MAX

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

Minimax algorithm. Example

MAX

MIN

MAX

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

CS 1571 Intro to AI

M. Hauskrecht
Minimax algorithm. Example
Minimax algorithm

function MINIMAX-DECISION(game) returns an operator
  for each op in OPERATORS(game) do
    VALUE(op) ← MINIMAX-VALUE(APPLY(op, game), game)
  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.

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

Complexity: $O(b^m)$
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

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

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

\[\geq 4\]
Alpha beta pruning. Example

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

4 ≤ 4
4 ≥ 6
Alpha beta pruning. Example

<table>
<thead>
<tr>
<th>MAX</th>
<th>MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

\( \geq 4 \)

\( \geq 6 \)

\( \geq 2 \)
Alpha beta pruning. Example

\[
\begin{array}{cccccccccccc}
4 & 3 & 6 & 2 & 2 & 1 & 9 & 5 & 3 & 1 & 5 & 4 & 7 & 5 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
4 & 3 & 6 & 2 & 2 & 1 & 9 & 5 & 3 & 1 & 5 & 4 & 7 & 5 \\
\end{array}
\]
Alpha beta pruning. Example

MAX

MIN

MAX

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

= 4

≥ 6

= 2

≤ 2
Alpha beta pruning. Example

MAX

MIN

MAX

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

4 = 6 ≥ 4 = 5 ≥ 2 ≤ 5 ≥ 7

CS 1571 Intro to AI
M. Hauskrecht
Alpha-beta pruning. Example

\[
\begin{align*}
\text{MAX} & \quad = 5 \\
\text{MIN} & \quad = 4 \\
\text{MAX} & \quad = 4 \\
& \quad \geq 6 \\
& \quad = 2 \\
& \quad \leq 2 \\
& \quad = 2 \\
& \quad \leq 5 \\
& \quad = 5 \\
& \quad \geq 7
\end{align*}
\]

\[= 6 \geq 4 = 5 \leq 5 = 7\]

\[\text{nodes that were never explored !!!}\]

---

Alpha-Beta pruning

**function** MAX-VALUE(state, game, α, β) returns the minimax value of state

**inputs:** state, current state in game

- game, game description
- α, the best score for MAX along the path to state
- β, the best score for MIN along the path to state

**if** GOAL-TEST(state) **then return** Eval(state)

**for each s in SUCCESSORS(state) do**

- α ← MAX(s, MIN-VALUE(s, game, α, β))

**if** α ≥ β **then return** β

**end return s**

**function** MIN-VALUE(state, game, α, β) returns the minimax value of state

**if** GOAL-TEST(state) **then return** Eval(state)

**for each s in SUCCESSORS(state) do**

\[β ← \text{Min}(β, MAX-VALUE(s, game, α, β))\]

**if** β ≤ α **then return** α

**end return** β
Using minimax value estimates

- Idea:
  - Cutoff the search tree before the terminal state is reached
  - Use imperfect estimate of the minimax value at the leaves
    - Evaluation function

MAX

MIN

Heuristic evaluation function

Cutoff level

Design of evaluation functions

- **Heuristic estimate** of the value for a sub-tree
- **Examples of a heuristic functions:**
  - **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

Heuristic estimates