Opponent Modeling and UCT

Suiteng Lu
Outline

- Poker
- Motivation
- MCTS in Poker
- Learning an Opponent Model
- Integrating an Opponent Model with MCTS
- Experiments and Results
Poker

- Card game between at least two players
- Objective is to win games (best card combination or only active player)
- Game include several betting rounds
- Player actions:
  - Call/Check
  - Bet/Raise
  - Fold
- Focus on Texas Holdem FL HU
  - 2 players
  - 4 rounds (Preflop, Flop, Turn, River)
  - Fixed betting/raising amount
Motivation

▪ Nash Equilibrium vs. Opponent Exploitation
  ▪ Nash-equilibria-strategy is oblivious to opponent mistakes
  ▪ Exploitive strategy is vulnerable to counter strategies

▪ Example: Rock-Paper-Scissors
  ▪ Optimal strategy: choose each option 1/3 of the time
  ▪ Exploitation: if opponent deviates from optimum, play a pure strategy in response

▪ Create a poker agent that can both

=> Integrating Opponent Models with Monte-Carlo Tree Search
Monte-Carlo Tree Search in Poker

- Every node represents a game state. Statistics of the node are:
  - **Value** of the game state. Average of the reward of all simulated games that visited this node
  - **Visit count** of the game state. Number of simulations in which this state was reached

- Start from the root node which is initially the only node in the tree

- Consists of repeating 4 steps
  - Selection
  - Expansion
  - Simulation
  - Backpropagation
Selection

- Exploitation vs. Exploration
- Use UCT to select the next child node which has the highest expected value

\[ k \in \arg\max_{i \in I} (v_i + C \times \sqrt{\frac{\ln n_p}{n_i}}) \]

- \( i \in I \): Set of nodes reachable from the current node \( p \)
- \( v_i \): Expected value of the node \( i \)
- \( n_i \): Visit count of the node \( i \)
- \( n_p \): Visit count of the parent node \( p \)
- \( C \): Coefficient that balances exploration and exploitation
Expansion

- When a leaf node is selected expand the tree by adding a new node
- For each simulated game the tree is expanded by one node
Simulation and Backpropagation

- Simulation from expanded node until end of the game
  - More realistic simulations will have significant effect on computed EV
- Roll-Out simulations in Poker
  - All remaining active players call or check until end of the game
- Update each tree node that was traversed during the game
  - Increase visit count
  - Modify values
Learning an Opponent Model

- MCTS in Selfplay produces an approximated Nash Equilibrium strategy
- Treats Opponent as rational players and oblivious to opponent mistakes
- Human players do not play a perfect rational strategy
- A tailored counter strategy will earn more profit than rational strategy

=> We need an accurate model of opponent strategy
Learning an Opponent Model

- Model is learned based on experience. Previous games of opponent as training data: \((i, p, a_i, c_p, S_{i-1})\)

- Model is used to estimate opponent behavior in unseen situations

- Learning task is to predict cards or action for opponent given the observed game state information: \(P(c_p|S_{i-1})\) \(P(a_i|S_{i-1}, c_p)\)

- More training data leads to more accurate models
Learning an Opponent Model

- **Approach**
  - Problem: Limited amount of training data
  - Remedy: Use a reasonable prior opponent model e.g. model of a rational player or model of a specific player type
  - If no training data available use prior as default
  - Adapt prior to opponent using a **differentiating function** as soon as more training data becomes available

- **Differentiating function**
  - Given collected training data for a player which reflects the player distribution $P(D_p|x)$
  - Create a data set consists of all collected training data and an equal amount of examples drawn from the prior distribution
  - Employ the relational probability tree algorithm *TILDE* on the data to learn the differentiating function
Relational Decision Trees (TILDE)

- Start with one leaf which contains all stored examples

- When it reaches a leaf node the tree may be expanded by candidate tests that partition the state space
  - Candidate tests describe a part of the game state e.g. sum_hole_cards
  - Choose the best candidate test that reduce variance among the two distribution sufficiently

- Internal nodes contain a test which is a conjunction of all literals in the path from root to this node

- Terminal nodes contain zero or more examples of the two distributions
• Learning a differentiating function for predicting cards

• Data set consists of examples 50% from prior and 50% from player distribution
Posterior Probability

- Use the differentiating function \( P(D_p|x) \) and the prior probability \( P(x|D_*) \) to compute the posterior probability \( P(x|D_p) \) by using Bayes-Rule

\[
P(x|D_p) = P(D_p|x) \cdot P(x)/P(D_p) \quad (1)
\]

\[
P(D_p) = P(D_*) = 1/2 \quad (2)
\]

\[
P(x) = P(D_*)P(x|D_*) + P(D_p)P(x|D_p) \quad (3)
\]

Substitute (2) and (3) into (1) gives:

\[
P(x|D_p) = \frac{P(x|D_*) \cdot P(D_p|x)}{1 - P(D_p|x)}
\]

- It is the probability that the opponent performs an action (or has the specific cards) given a certain game state
Integrating Opponent Model with MCTS

- Use opponent card prediction to determine the reward at end of each iteration

- Use opponent action probabilities on opponent nodes in the selection phase

```plaintext
Data: game_state \( S_{i-1} \)
Data: root_node
Data: card_probabilities \( P(c_p|S_{i-1}) \forall c_p \)
Result: best_move

while (has_time) do
    current_node ← root_node
    Sample \( c_p \) from card_probabilities
    while (current_node ∈ \( T \)) do
        last_node ← current_node
        if current_node ≠ opponent_node then
            \( P ← UCT\_Prob \forall a_i \)
        else
            \( P ← P(a_i|S_{i-1},c_p) \forall a_i \)
        current_node ← Select(current_node, \( P \))
    end
    last_node = Expand(last_node)
    R ← Play_simulated_game(last_node)
    while (current_node ∈ \( T \)) do
        current_node ← current_node.parent
        Backpropagation(current_node, \( R \))
    end
end
return best_move = \( \text{argmax}_{N \in N_c}(root\_node) \)
```
Experiments and Results

- **Experiment**

  - Compare standard MCTS and MCTS integrated with opponent models in heads up game against 2 different bots
  
    - **ACE1**: Simple rule-based bot, straightforward, actions based on numerical rankings of the cards
    
    - **POKI**: Formula-based bot, effective hand strength as main input. comparable to skilled human players
Experiment Setup

- 5000 collected training data for each opponent bot
- Uniform distribution as prior
- 1000 iterations at each decision node
- UCT coefficient $C = 2$
- Candidate tests used by TILDE
  - Phase, number of bets,
  - Previously executed actions by players
  - Pot Odds, Pot ratio
  - Numerical Rankings of hands, sum of ranks
  - Suits, pairs of private cards
Results

- Both runs beats ACE1 with large amount
- Agent with opponent model shows great increase in aggression due to the simple strategy of ACE1
- Poor performance against POKI, the reasons can be low number of iterations, or wrong choice of parameters
- Overall large improvement after using integrated opponent models against both opponent bots

<table>
<thead>
<tr>
<th></th>
<th>sf</th>
<th>p-agr</th>
<th>agr</th>
<th>sb/h</th>
<th>games</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE1</td>
<td>0.64</td>
<td>0.5</td>
<td>4.7</td>
<td>1.09</td>
<td>10010</td>
</tr>
<tr>
<td>ACE1-M</td>
<td>0.47</td>
<td>7311</td>
<td>8.3</td>
<td>1.67</td>
<td>10010</td>
</tr>
<tr>
<td>POKI</td>
<td>0.945</td>
<td>0.7</td>
<td>1.9</td>
<td>-0.50</td>
<td>10304</td>
</tr>
<tr>
<td>POKI-M</td>
<td>0.916</td>
<td>0.2</td>
<td>0.8</td>
<td>0.06</td>
<td>10310</td>
</tr>
</tbody>
</table>
Questions

Any Questions?
References

- Integrating Opponent Models with Monte-Carlo Tree Search in Poker, M. Pellier, G. Gerritsen, G. M. J.-B. Chaslot, 2010