Superhuman AI for Multiplayer Poker

Theoretical and practical challenges of multiplayer games

- Prior methods attempted to approximate a Nash equilibrium strategy, rather than detecting and exploiting weaknesses in the opponent.

- *Nash equilibrium* - list of strategies, one for each player, such that no player can improve by deviating to a different strategy.

- Nash equilibria are guaranteed to exist for all finite games and some infinite games too.

- For a two-player zero sum game, if a player plays a Nash equilibria strategy, then he is guaranteed not to lose in expectation, no matter the opponent.

  For this reason, solving a two-player zero sum game is tantamount to finding the nash equilibrium.

- Blend nash equilibrium strategy with exploitation - start by playing a nash equilibrium strategy, then tailor the strategy to the observed weaknesses of your opponent's strategy.

  Ex: The optimal strategy in row-sham-bow is to play each action randomly.
  However, if you observe your opponent only plays paper, then the better strategy would be to only play rock.

- Oftentimes, playing a blended strategy also opens the player up to exploitation themselves.

  Key Note: Pluribus plays a fixed strategy that doesn't adapt to the observed tendencies of the opponents.

- Solving for or approximating a Nash equilibrium strategy is computationally difficult.

  And, solving for the NE for multiplayer poker is so difficult and computationally expensive to the point of practical impossibility.

- A player solving for the NE in a multiplayer game might not be optimal anyways.

  Ex: Lemonade Stand Game - each player can compute an individual NE that when combined with other players actions to create a joint strategy that is suboptimal.

  Two player zero-sum games are an exception to this rule, two players can compute NEs independently that create a joint strategy that is still a NE.

- In the absence of the ability or incentive to compute a NE for six-player poker, we instead seek to create an AI capable of superhuman performance against humans.

  High empirical performance is achieved despite a lack of theoretical guarantees.

Description of Pluribus

- Pluribus uses self play (with no prior input, human or otherwise) to compute action values.
- Before play, Pluribus computes an offline "blueprint" strategy.
  (I assume this is something like a model?)
- During play, Pluribus improves upon the blueprint strategy by searching the game tree with the root being the current state it is in.
  (This sounds like a form of MCTS to me?)

Abstraction for large imperfect-information games

- Two key abstractions:
  - Action abstraction
    - The number of actions are reduced to between 1 and 14 depending on the situation.
    - Off-tree actions, not seen during offline planning, can be dealt with by the search algorithm.

      Ex: What do we do if an opponent bets $150 when we have only seen $100 and $200 when we constructed our blueprint.
  - Information abstraction
    - Similar decision points are bucketed together and treated identically.
      (Sounds like tile coding?)
    - When online, Pluribus only uses information abstraction to reason about situations on future betting rounds.

Self-play through improved Monte Carlo counterfactual regret minimization

- *Counterfactual regret minimization* (CFR) - iterative self-play algorithm in which the AI starts with a random policy and improves by playing older versions of itself.

Guaranteed that the average strategy over all iterations converges to a NE, for all two-player zero-sum games.

- Blueprint strategy was computed using CFR.
- *Monte Carlo CFR* (MCCFR) - samples actions in the game tree rather than doing a full update through the entire game tree for each iteration.
  (Sounds like the difference between MC and DP?)
- One player's strategy is improved per iteration.

  This player is designated as the "traverser".
- A complete poker hand is simulated using the players' policies.
- The traverser then compares the actions it took versus alternative actions that it could've made.

  Because we have a complete model of the system, we can see what would have happened had a different action been taken.

  This is called counterfactual reasoning.
- *Counterfactual regret* - the difference between what the traverser would have received for choosing an action versus what the traverser actually achieved.
  (in expectation).

  Actions with higher regret are chosen over those with lower regret.
- *Linear CFR* - form of CFR that adds a discounting weight to the regret contributions. This way the contribution of the initial iterations diminishes as more experience is gained, at a rate of $1/T$.

  In practice this allows the strategy to improve more quickly while still maintaining a similar worst-case bound on total regret.
- Actions with extremely negative regret are not explored in 95% of iterations.
- Compute and storage configuration:
  - The Blueprint strategy was computed in 8 days.
  - 64-core server for 12,400 CPU core hours and 512 GB of memory.
  - Approx. $144 to produce using cloud computing services.
  - Pluribus was capable of live play on a 128 GB memory machine.
  - During live play, a compressed form of the blueprint strategy was kept in memory.

Depth-limited search in imperfect-information games

- Pluribus only plays according to the Blueprint for the first round of betting.
  (because the action space is small enough to be approximated accurately enough).

  Within a bound, off-tree opponent bet sizes are rounded to the Blueprint's bet sizes.

  Exception to this is if an opponent uses a bet size that is sufficiently unfamiliar to the Blueprint's bet sizing. In which case, search is used.

- After the initial betting round, Pluribus uses search to develop a finer-grained strategy.

- Real-time search has been highly effective at perfect-information games.
  (backgammon, chess, go, etc.)

- *Subgame* - part of the game in which search is being conducted.

- Traditional search applied to imperfect-information games is flawed.

- Challenge #1 - Leaf nodes do not have a fixed value in imperfect-information games because they depend on the strategy that the searcher chooses in the subgame.

  Ex: Rock-paper-scissors where Player 1's action is hidden from Player 2's.
  If Player 2 plays the NE strategy of picking each action equally, then Player 1 will calculate that each action has an expected value of zero.
  In which case, a strategy of always choosing rock would be a valid optimal policy for Player 1.
  However we know this is not true.

- Potential Solution #1 - make the value of a leaf node conditional only on the belief distribution of both players at that point in the game.

  Ex: Two-player poker AI DeepStack

  Drawback - It's expensive to solve a huge number of subgames conditioned on player beliefs.
  This gets worse as the number of players and hidden information scales.

- Potential Solution #2 - only perform search when the depth limit reaches the end of the game, thereby eliminating non-terminal leaf nodes.

  Ex: Two-player poker AI Libratus

  Drawback - when more than two-players are considered, always expanding to the end of the game becomes computationally prohibitive.

- Potential Solution #3 - searcher explicitly considers that any or all players may shift to different strategies beyond the leaf nodes of a subgame, rather than assuming all players play according to a single fixed strategy.

  This choosing of different strategies at leaf nodes prevents leaf nodes from having a single fixed value.

  Once a leaf node is encountered, each player chooses from k available strategies to play for the remainder of the game.

  Each set of available strategies is specialized to the player.

  Ex: Pluribus.

  Pluribus uses $k = 4$, where the strategies are:
  - Precomputed blueprint strategy
  - Precomputed blueprint strategy with a bias toward folding.
  - Precomputed blueprint strategy with a bias toward calling.
  - Precomputed blueprint strategy with a bias toward raising.

  This creates a more balanced strategy because the AI must account for a shift in its opponent's strategy.

  Ex: Going back to the row-sham-bow example, if we only play rock, then our opponent could shift their strategy to only playing paper. This solution accounts for this variability of strategy.

- Challenge #2 - A player's optimal strategy in a given situation depends on its strategy in every other situation.

  Ex: If a player only bets in a given situation when they have the best hand, then its opponent's response would be to always fold.

- Solution - Compute a balanced strategy:
  - Track the probability of reaching the current position with each possible hand, according to its strategy.
  - Balance its strategy in this position for each possible hand.
  - Execute an action for its given hand.

- Pluribus used two different forms of CFR to compute its strategy in the subgame depending on size and part of the subgame:
  - For large subgames or for early game positions:
    - Linear MCCFR

- Otherwise:
    - Optimized vector-based Linear CFR that only samples chance events. (i.e. board cards).