2023-Schmid

Student of Games: A unified learning algorithm for both perfect and imperfect information games

---

- **Date**: 2023-11-15
- Link
- **Authors:**
  - Schmid, Martin
  - Moravcik, Matej
  - Burch, Neil
  - Kadlec, Rudolf
  - Davidson, Josh
  - Waugh, Kevin
  - Bard, Nolan
  - Timbers, Finbarr
  - Lanctot, Marc
  - Holland, Zacharias
  - Davoodi, Elnaz
  - Christianson, Alden
  - Bowling, Michael
- **Cites:**
- **Cited by:**
- **Keywords:**
- **Collections:** #related-to-poker
- **Status:** #in-progress

---

0. - Abstract

-
  - Two types of problems with two variants of solutions:
  - Perfect information games - search and learning.
  - Imperfect information (poker variants) - game-theoretic reasoning and learning.

- *Student of Games* (SoG) - general-purpose algorithm that unifies previous approaches, combining guided search, self-play learning, and game-theoretic reasoning.
  - Shown to produce strong empirical results across perfect and imperfect info games.

1. - Introduction

- For much of AI history, the focus has been to develop algorithms that work for a specific game.
  - Eg. DeepBlue can play chess but not checkers.

- Some efforts have been made for general perfect information game algorithms.
  - Eg. DeepZero can play Go and Chess.

- No unification effort has been made for imperfect information games, and, hereto, no connection has been drawn between perfect and imperfect game solving algorithms.

- SoG uses the following tools to unify perfect and imperfect information game algorithms:
  - *Growing-tree counterfactual regret minimization* (GT-CFR) - anytime local search that builds subgames nonuniformly, expanding the tree toward the most relevant future states while iteratively refining values and policies.

  - *Sound self-play* - learning procedure that trains value-and-policy networks using both game outcomes and recursive sub-searches applied to situations that arose in previous searches.

1.1 - Background and terminology

- This paper uses the Factored-Observation Stochastic Games (FOSG) formalism.

- Definitions:
  - States - $w \in \mathcal{W}$, let w^{init} be the starting state.

  - Actions - $a \in \mathcal{A}$, actions transition one state to the next, until a terminal state is reached.
    - Let $\mathcal{A}(w) \subseteq \mathcal{A}$ be the set of valid actions at state w.

  - Decision node - player, $\mathcal{P}(w)$ makes an action.

  - History - $h \in \mathcal{H}$, sequence of states and actions.
    - $h' \sqsubseteq h$ be a prefix history (subsequence).

- Terminal history - $z \subset H$, a history that ends in a terminal state, each player receives a utility $u_i(z)$.

- Information state - $s_i \in \mathcal{S}_i$, set of histories that are indistinguishable to player i.

- Policy - $\pi_i : \mathcal{S}_i \rightarrow \Delta(\mathcal{A})$, policy for player i at info state $\mathcal{S}_i$ is a probability distribution over available actions.

- Observations - every time a player takes an action a in the state w changing the game state to w', they receive a public, $\mathcal{O}_{\text{pub}}(w, a, w')$, and private observation, $\mathcal{O}_{\text{priv}(i)}(w, a, w')$.

  - In perfect information games:
    - $\mathcal{O}_{\text{priv}(i)}(w, a, w') = \mathcal{O}_{\text{pub}}(w, a, w')$.
    - Game dynamics only depend on player actions, $\mathcal{T}(w, a) = \mathcal{T}[w, a_{\mathcal{P}(w)}]$.

  - In imperfect information games:
    - Information asymmetry - players receive information that their opponent's are not privy to.

- Public state - $s_{\text{pub}}(h) \in \mathcal{S}_{\text{pub}}$, sequence of public observations encountered along the history h.
  - Eg. in poker, all bets and antes plus all cards dealt on the board.
  - Let $\mathcal{S}_i(s_{\text{pub}})$ be the set of possible info states for player i given the public state s_{pub}.
    - i.e. same public states but different private observations.
    - e.g. we face the same bet sequence and flop cards, but with different hole cards.

- Public belief state - $\beta = (s_{\text{pub}}, r)$ where $r \in \Delta[\mathcal{S}_1(s_{\text{pub}})] \times \Delta[\mathcal{S}_2(s_{\text{pub}})]$ is the range which defines a pair of distributions over possible info states for both players.
  - Player's level of belief in which state they are in.
  - E.g. if their opponent goes all in, then they should believe they're more likely to be in a state in which their opponent has a stronger hand than a weaker one.

- Policy profile - $\pi(\pi_1, \pi_2)$, pair of player policies.

- Expected utility - $u_i(\pi_1, \pi_2)$.

- Best response - π_i^b, any policy that maximizes the expected utility against the opponent's policy, π_{-i}.
  - $\pi_i^b \in \{\pi_i | u(\pi_i, \pi_{-i}) = \max_{\pi_i'} u_i(\pi_i', \pi_{-i})\}$

- Nash equilibrium - if π_1 is a best response to π_2 and vice versa.
  - For two-player zero-sum games, NEs maximize the players' worst case utility guarantees.

- Approximate equilibrium - $u_i(\pi_i^b, \pi_{-i}) - u_i(\pi_i, \pi_{-i}) \leq \epsilon$ for all players i.

- Strategy exploitability - how much, on average, a player will lose against a best response opponent, relative to if they would've played the NE strategy.
  - $\text{Exploitability}(\pi) = [\max_{\pi_{1'}} u_1(\pi_{1'}, \pi_2) + \max_{\pi_{2'}} u_2(\pi_1, \pi_{2'})]/2$

1.2 - Tree search and machine learning

- *Minimax* - perfect information two-player zero-sum game algorithm:
  - Depth-limited search is performed at the current world state $w_{\mathcal{D}}$.
  - A heuristic evaluation function is used to estimate the value of states beyond the depth limit, $h(w_{t+d})$.
  - These values are then backed-up using game-theoretic reasoning.

- Minimax provided AI's first major milestones.
  - e.g. IBM's super-human DeepBlue chess program.

- *Monte Carlo tree search* (MCTS) - evolution of minimax algorithm for more complex games by building game trees via simulations:
  - Starts with an empty tree rooted at w_t.
  - Expands the tree by adding simulated trajectories.
  - Estimates values from rollouts to the end of the game.

- MCTS allowed for substantially stronger play in Go and other games; however, heuristics and domain knowledge were still required.

- *AlphaGo* - replaced heuristics and domain knowledge with value functions, approximated by deep neural networks, and policies learned via human export data and improved via self-play.

- *AlphaGo* achieved super-human level play in Go.

- *AlphaGo Zero* (AlphaZero)- removed the need for initial training from human expert play and any Go-specific features which allowed it to also achieve SotA performance in Shogi and chess as well.

- SoG combines search and learning from self-play using minimal domain knowledge and uses counterfactual regret minimization for sound reasoning in imperfect information games.

1.3 - Game-theoretic reasoning and counterfactual regret minimization

- *Game theoretic reasoning* - players must choose their strategy so as to not reveal their hidden information.
  - E.g. a poker player should not play so predictably that their opponent can guess their cards.

  - E.g. you should not always choose rock in row-sham-bow.

- Game theoretic reasoning arises out of player's computing approximate minimax-optimal strategies.

- *Counterfactual regret minimization* (CFR) - iterative, self-play algorithm for computing approximately optimal strategies:

  - Produces policy iterates, $\pi_i^t(s, \cdot) \in \Delta(\mathcal{A})$ for each player i for each info state s such that the player's long-term average regret is minimized.

  - Average policy over all T iterations, $\bar{\pi}^T$ converges to ϵ-NE at a rate of $O(1/\sqrt{T})$.

  - *Counterfactual value* - $v_i^t(s, a)$, value of playing a in state s at time t for iteration i.

  - *Counterfactual regret* - regret of playing a in state s at time t for iteration i:
    $r^t(s, a) = v_i^t(s, a) - \sum_{a \in \mathcal{A}(s)} \pi^t(s, a) v_i^t(s, a)$

  - *Cumulative regret* - $R^T(s, a) = \sum_{t=1}^T r^t(s, a)$

  - *Regret matching* - policy update rule using cumulative regrets:

$$\pi^{t+1}(s, \cdot) = \frac{[R^t(s, \cdot)]^+}{\sum_a [R^t(s, a)]^+}$$

    ,

- *CFR+* - modification to the cumulative regret formulation in CFR:
  - $Q^t = [Q^{t-1}(s, a) + r^t(s, a)]^+$
  - $\pi^{t+1}(s, a) = Q^t(s, a) / \sum_b Q^t(s, b)$

- *Beliefs* - each player's probability of reaching each information state under their policy (called their range).
  - Necessary for CFR computation.

1.4 - Imperfect information search, decomposition, and re-solving

- Traditional CFR:
  - Used as a game-solving engine, computing entire policies via self-play.
  - Each iteration traverses the entire game tree, starting at the game root.

- *Subgame decomposition* - process by which CFR is used to compute a policy starting at an arbitrary initial public state and ending at a give depth:

  - Computes a policy for a part of the game up to a depth $d > 0$.

  - Can start at any public state.

  - An oracle provides the counterfactual value each player would receive at depth d.

  - Paired with a belief distribution r over initial information states, $s \in \mathcal{S}_i(s_{\text{sub}})$.
    - Note: for perfect information games, the probability distribution for both players is a singlet.

- Subgame decomposition:
  - Has been critical for recent AI advancements including HUNL.
  - Analogous to search in perfect information games and traditional Bellman-style bootstrapping.

- *Counterfactual value network* (CVN) - encodes the value function:

$$v_\theta(\beta) = \{v_i(s_i)\}_{s_i \in \mathcal{S}_i(s_{\text{pub}}), i \in \{1,2\}}$$

  where:
  - β are the player's beliefs over information states for the public info at s_{pub}.
  - θ are parameters for the network.

- CVN can be used in place of the oracle in subgame decomposition.

- *Safe re-solving* - technique that generates subgame policies from summary information computed by a previous approximate solution.

  Two bits of information needed:
  - The player's range
  - The opponent's counterfactual values.

- Safe re-solving exploitability guarantees:
  - Constructs an auxiliary subgame with specific constraints.
  - Subgame policies are generated in such a way to preserve exploitability guarantees.
  - This allows the computed subgame policies to be substituted for the original policies.

- *Continual re-solving* - algorithm that performs safe re-solving at every decision point.
  - Analogous to classical game search but adapted to suit imperfect information games.

1.5 - Related Work

- SoG compared to AlphaGo:
  - Similarities:
    - Uses search and deep NN learning.
  - Differences:

        - Search is sound for imperfect information games.

- SoG compared to DeepStack:
    - Similarities:
        - Uses search and deep NN learning.
        - Uses game theoretic reasoning and imperfect info search.
    - Differences:
        - Uses less domain knowledge.
        - Use of self-play rather than poker heuristics.

- SoG is similar to Recurrent Belief-based Learning but uses safe continual re-solving and sound self-play, regardless of the algorithm used in training.

- There have been many other attempts at search in imperfect information games.
  (See paper for more details).

1.6 - Descriptions of challenge domains

- See paper for details about Chess, Go, HUNL, and Scotland Yard.

2. - Results

2.1 - SoG: Algorithm summary

- SoG trains an agent via sound self-play.

- To make a decision:
    - GT-CFR search with a CVPN is used to generate a policy for the current state.
    - An action is then sampled from this policy.

- CT-CFR search iteration:
    - *Regret update phase* - runs public tree CFR updates on the current tree.
    - *Expansion phase* - adds new public stats using simulation-based expansion trajectories.

- *Search queries* - public belief states that were queried by the CVPN during the CT-CFR regret update phase.

- Data used for updated the value and policy networks during training:
    - Search queries.
        - Must be solved to compute counterfactual value targets for updating the value network. <br/
    - Full-game trajectories from self-play games.
        - Provide targets for updating the policy network.

- In practice, self-play and training happen in parallel:
    - Actors generate self-play data (solve queries).
    - Trainers learn new networks and periodically update the actors.

Sections 2.2. - 2.5

- See paper for results graphs and tables.

3. - Discussion

- SoG overview:
    - Purpose - unifying algorithm that combines search, learning, and game-theoretic reasoning.

    - Main components:
        1. GT-CFR
        2. Sound self-play using CVPNs.

    - Strong theoretical and empirical backing.

4. - Materials and Methods

4.1 - Counterfactual value-and-policy networks

- CVPN with parameters θ, represents a function, $f_\theta(\beta) = (\mathbf{v}, \mathbf{p})$.
    - Where:
        - $\mathbf{v}$ - counterfactual values, one per information state per player.
        - $\mathbf{p}$ - prior policies, one per information sate for the acting player.

4.2 - Search via GT-CFR

- Algorithm:

    - Initial tree $\mathcal{L}^0$ contains β and its child public states.

    - Each iteration t:
        - *Regret update phase* - runs $\frac{1}{c}$ public tree CFR updates on the tree, $\mathcal{L}^t$.

- *Expansion phase* - adds public states to $\mathcal{L}^t$ using simulation-based expansion trajectories, producing $\mathcal{L}^{t+1}$. Where s is defined as the total expansion simulations.

- Simulation details:
  - Search statistics are maintained over information states, accumulated over all expansion phases.

  - Simulation is initialized by sampling:
    - An info state s_i from the given beliefs β_{root}.
    - A world state w_{root} and associated history h_{root} is sampled from s_i.

  - Actions are selected according to a mixed policy:

$$\pi_{\text{select}}[s_i(h)] = \frac{1}{2}\pi_{\text{PUCT}}[s_i(h)] + \frac{1}{2}\pi_{\text{CFR}}[s_i(h)]$$

    where:

    - π_{PUCT} - takes into account learned values by using counterfactual values $v_i(s_i, a)$ normalized by the sum of the opponent's reach probability at s_i. (See PUCT paper for more info).

    - π_{CFR} - takes into account the currently active policy from search by averaging CFR's policy at $s_i(h)$.

  - The simulation terminates once a public state outside the tree is encountered, $s_{\text{pub}} \notin \mathcal{L}$.
    - s_{pub} is added to the tree.
    - Visit counts are updated for nodes visited during the simulation.

  - Optionally, only the top k actions can be considered during simulation for a given decision.
    - This allows for speed-ups when we expect our optimal policy to be relatively deterministic, e.g. perfect information games.

4.3 - Modified continual re-solving

- *Problem*:
  - Prior implementations of re-solving have taken advantage of the fact that the current public state, s_{pub}, was included in the previous state's, $s_{\text{pub}}^{\text{prev}}$, search tree.

  - Recall that all we need to reconstruct the strategy are the current player's range and the opponent's strategy.

  - If our current state was not included in the search tree for the last state, then we do not have these values.

- *Solution*:
  - Start the re-solving process in the state closest to the current state that is included in the previous search tree.

  - Initialize the search tree with a single branch from this state to the current state.

  - Only grow the tree under the current state.

  - To compute the opponent's range we use a gadget that mixes the opponent's range from the previous search, $\alpha r + (1-a)r^{\text{prev}}$.
    - In practice, $\alpha = 0.5$ improves performance.

4.4 - Performance guarantees for continual re-solving

- See supplementary text for proofs of these theorems.

- **Theorem 1** - The regret at iteration T for player i is bounded by,

$$R_i^{T,\text{ full}} \leq \sum_{t=1}^{T} |\mathcal{F}(\mathcal{L}^t)|\, \epsilon + \sum_{s_{\text{pub}} \in \mathcal{N}(\mathcal{L}^T)} |\mathcal{S}_i(s_{\text{pub}})|\, U\sqrt{AT}$$

  where:
  - $\mathcal{N}(\mathcal{L})$ - interior of the tree, all non-leaf, non-terminal public states where GT-CFR generates a policy.

  - $\mathcal{F}(\mathcal{L})$ - frontier of the tree, all nonterminal leaves where GT-CFR uses ϵ-noisy estimates of counterfactual values.

  - U - maximum difference in counterfactual value between any two strategies.

  - A - maximum number of actions at any information state.

- This can be thought of as the gap in performance between GT-CFR iterations and the highest-value strategy.

- **Theorem 2** - The exploitability of the final strategy is bounded by,

$$(5D+2)\left(F\epsilon + NU\sqrt{A/T}\right)$$

  where:
  - D - number of re-solving steps.
  - T - number of iterations of GT-CFR per re-solving step.
  - N - maximum interior size of the search tree.
  - F - maximum frontier size of the search tree.
  - A - maximum number of actions at any information state.

- U - maximum difference in values between any two strategies.

- General properties of the exploitability of re-solving with GT-CFR:
  - Exploitability decreases with more computation time, T.
  - Exploitability decreases with improved value function error, ϵ.
  - Exploitability only increases linearly with game length, D.

- Computational complexity of re-solving with GT-CFR:
  - In general - with k children - $O(kT^2)$ states visited and CVPN calls, per re-solving step.
  - For perfect information games - $O(T)$

4.5 - Data generation via sound self-play

- SoG generates episodes of data that are then used to train the CVPN.

- Searches performed at different public states should be consistent with both the CVPN and with searches made at previous states.

4.6 - Training process

- See Figure 8 in the paper.

4.7 - Query collection

- Once the episodes are completed, the queries saved during search are used to train the CVPN.

- An FIFO buffer is used to train the CVPN asynchronously.

- See Figure 8 for more details.

4.8 - Computing training targets

- Policy targets are assembled from histories reached in self-play.

- Value targets are computed in two ways:

  1. The outcome of the game is used as a $TD(1)$ value target for states along the main line of episodes.

  2. Via bootstrapping using GT-CFR on subgames rooted at input queries.
     - This can be thought of as a policy improvement operator.

4.9 - Recursive queries

- *Recursive query* - A query that is generated by the solver itself while running GT-CFR on another query.

- Advantage - produce more reasonable answers for the leaves in a search, not just those on the self-play lines.

- To ensure the buffer isn't over run by recursive queries, the probability of generating them is low, between 0.1 and 0.2 in practice.

4.10 - Consistency of training process

- Training converges to optimal values asymptotically as $T \to \infty$.