Counterfactual Regret Minimization Algorithms in Imperfect Information Games

Student Name: **Yosef Berezovskiy**

Supervisor Name: **Dr Trehan Amitabh**

Presentation Outline

- Overview
  - Games and Research
  - Solutions in Games
  - Perfect vs. Imperfect Information Games
- Motivation
  - Poker as an Imperfect Information Game
  - Solutions to Poker
- Counterfactual Regret Minimization
  - Research Question
- Deliverables
- Solution demonstration
- Questions and Answers

Overview: Games and Research

- Games are long standing testbeds in computer science
  - Tic-Tac-Toe playing computer developed in 1952 created to showcase a computer's ability to make decisions

- Recent advancements surpassed human abilities
  - Deep Blue winning over chess grandmaster, AlphaGo beating a human professional

- Theoretical game solving

Overview: Solutions in Games

- Game theoretic solution
  - Nash Equilibrium

- Game theory applications extend beyonds games
  - Economics
  - Cybersecurity
  - Politics

Overview: Perfect vs. Imperfect Information Games

- Perfect information: all players have complete knowledge of the game state
  - Many existing solutions

- Imperfect information: hidden information from one or more players
  - Present a harder challenge
  - More realistic reflection of the real-world
  - Higher research value

Motivation: Poker as an Imperfect Information Game

- Hidden information (opponent's cards) – Element of chance (card dealing) – Bluffing and strategic decision-making

- Large number of possible game states – Complex decision trees – Need for probabilistic reasoning

- Real-world relevance – Models decision-making under uncertainty and bluffing

Motivation: Solutions to Poker

- Early approaches – linear programming (LP) and sequence-form linear programming (SFLP)
  - LP had exponential space complexity
    - Solved only simplest forms of IIGS, like Kuhn Poker
  - SFLP had quadratic space complexity
    - Solved closer to competitive variants of Poker like Rhode Island Holdem

- Annual Computer Poker Competition

- Counterfactual Regret Minimization - Most successful approach to date – Solved Heads-up Limit Hold'em in 2015

Counterfactual Regret Minimization

- Based on regret matching algorithm
- Iterative algorithm
- Minimizes counterfactual regret in each information set by traversing game tree recursively
  - Decomposes the task of finding equilibrium into smaller, manageable subproblems
- Achieves state-of-the-art results in solving poker and other two player zero sum extensive form games

Counterfactual Regret Minimization: Research Question

- Multiple variants of the algorithm exist:
  - Monte Carlo Counterfactual Regret Minimization (MCCFR)
  - Discounted Counterfactual Regret Minimization
  - CFR+
  - Predictive CFR+

- How do different CFR variants differ in terms of convergence speed, and what are their relative performances when implemented and analyzed comparatively?

- To contribute by:
  - Validation of theoretical claims
  - Scalability assessment
  - Evaluating algorithmic modification impact

Deliverables

1. Implementation of multiple CFR variants (original CFR, DCFR, CFR+, MCCFR)

2. Validation to confirm the effectiveness of the implemented algorithms

3. Select and implement benchmark game environments

4. Comparative analysis of CFR variants
   - Convergence analysis
   - Scalability analysis

Deliverables: Implementation of CFR variants

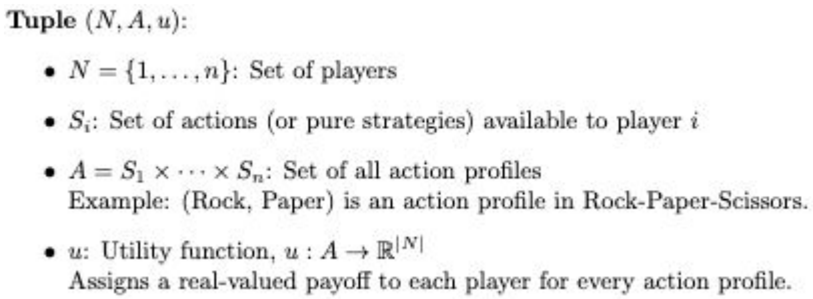
- Previous work in the field lacked mathematical framework to unify the definitions of the algorithms

- Hence why, implementation begins by establishing theoretical framework to:
  - Define foundations between CFR variants
  - Contextualise modifications and facilitating comparisons
  - Verify technical implementation adheres to theoretical specification

Deliverables: Implementation of CFR variants

- Establishing theoretical framework:

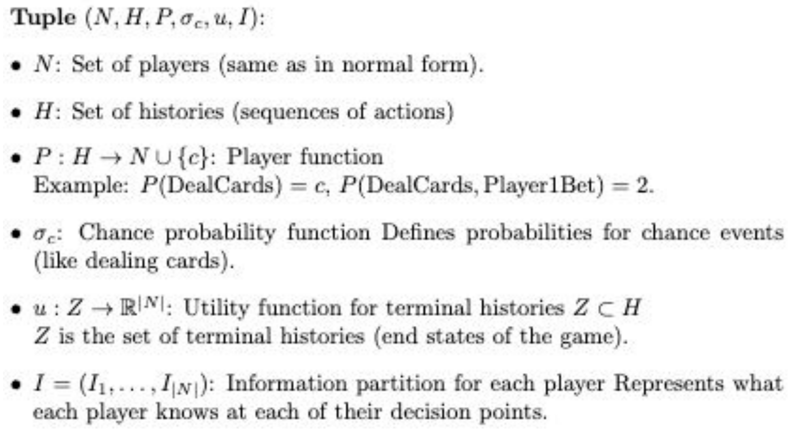
Normal form game:



Tuple (N, A, u):

- $N = \{1, \ldots, n\}$: Set of players
- S_i: Set of actions (or pure strategies) available to player i
- $A = S_1 \times \cdots \times S_n$: Set of all action profiles
  Example: (Rock, Paper) is an action profile in Rock-Paper-Scissors.
- u: Utility function, $u : A \to \mathbb{R}^{|N|}$
  Assigns a real-valued payoff to each player for every action profile.

Extensive form game:



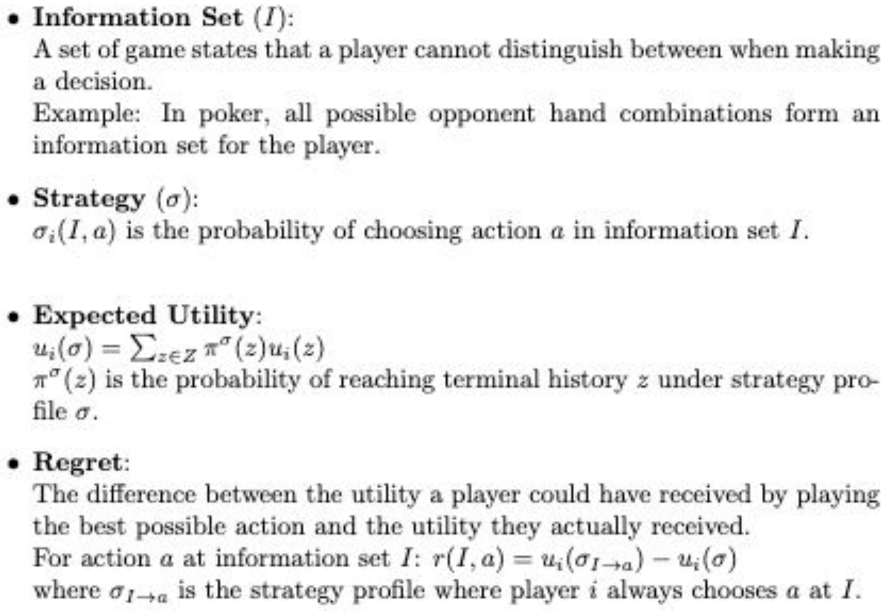
Tuple $(N, H, P, \sigma_c, u, I)$:

- N: Set of players (same as in normal form).
- H: Set of histories (sequences of actions)
- $P : H \to N \cup \{c\}$: Player function
  Example: $P(\text{DealCards}) = c$, $P(\text{DealCards}, \text{Player1Bet}) = 2$.
- σ_c: Chance probability function Defines probabilities for chance events (like dealing cards).
- $u : Z \to \mathbb{R}^{|N|}$: Utility function for terminal histories $Z \subset H$
  Z is the set of terminal histories (end states of the game).
- $I = (I_1, \ldots, I_{|N|})$: Information partition for each player Represents what each player knows at each of their decision points.

Deliverables: Implementation of CFR variants

- Establishing theoretical framework:

Other key concepts:



- **Information Set** (I):
  A set of game states that a player cannot distinguish between when making a decision.
  Example: In poker, all possible opponent hand combinations form an information set for the player.

- **Strategy** (σ):
  $\sigma_i(I, a)$ is the probability of choosing action a in information set I.

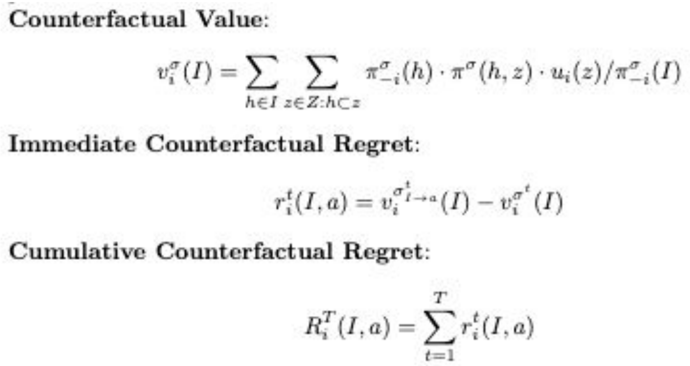
- **Expected Utility**:
  $u_i(\sigma) = \sum_{z \in Z} \pi^\sigma(z) u_i(z)$
  $\pi^\sigma(z)$ is the probability of reaching terminal history z under strategy profile σ.

- **Regret**:
  The difference between the utility a player could have received by playing the best possible action and the utility they actually received.
  For action a at information set I: $r(I, a) = u_i(\sigma_{I \to a}) - u_i(\sigma)$
  where $\sigma_{I \to a}$ is the strategy profile where player i always chooses a at I.

Deliverables: Implementation of CFR variants

- Establishing theoretical framework:

Vanilla CFR:



Counterfactual Value:

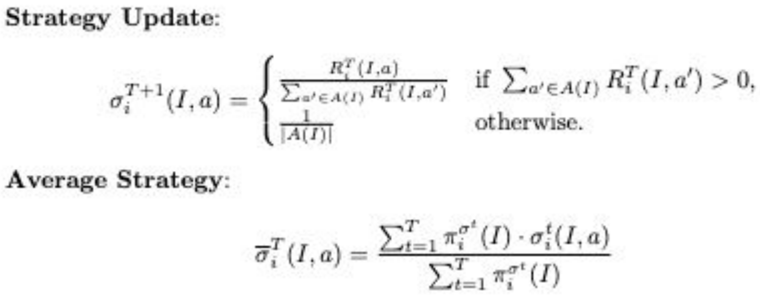
$$v_i^\sigma(I) = \sum_{h \in I} \sum_{z \in Z : h \subset z} \pi_{-i}^\sigma(h) \cdot \pi^\sigma(h, z) \cdot u_i(z) / \pi_{-i}^\sigma(I)$$

Immediate Counterfactual Regret:

$$r_i^t(I, a) = v_i^{\sigma^t_{I \to a}}(I) - v_i^{\sigma^t}(I)$$

Cumulative Counterfactual Regret:

$$R_i^T(I, a) = \sum_{t=1}^{T} r_i^t(I, a)$$



Strategy Update:

$$\sigma_i^{T+1}(I, a) = \begin{cases} \frac{R_i^T(I,a)}{\sum_{a' \in A(I)} R_i^T(I,a')} & \text{if } \sum_{a' \in A(I)} R_i^T(I, a') > 0, \\ \frac{1}{|A(I)|} & \text{otherwise.} \end{cases}$$

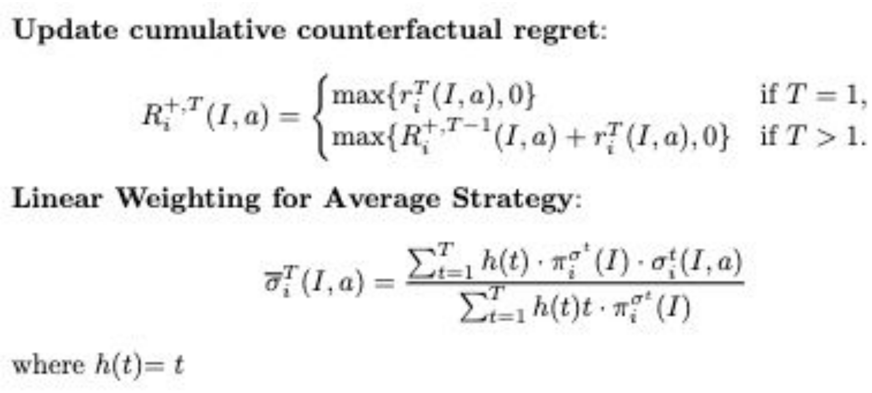
Average Strategy:

$$\overline{\sigma}_i^T(I, a) = \frac{\sum_{t=1}^T \pi_i^{\sigma^t}(I) \cdot \sigma_i^t(I, a)}{\sum_{t=1}^T \pi_i^{\sigma^t}(I)}$$

Deliverables: Implementation of CFR variants

- Establishing theoretical framework:

CFR+:



Update cumulative counterfactual regret:

$$R_i^{+,T}(I,a) = \begin{cases} \max\{r_i^T(I,a), 0\} & \text{if } T = 1, \\ \max\{R_i^{+,T-1}(I,a) + r_i^T(I,a), 0\} & \text{if } T > 1. \end{cases}$$

Linear Weighting for Average Strategy:

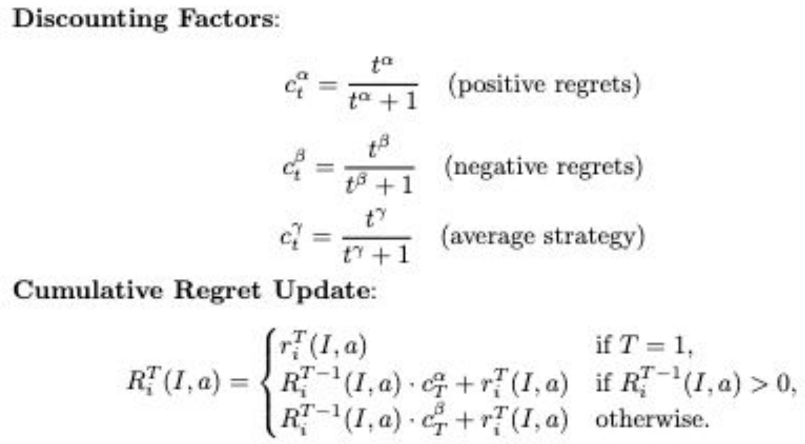
$$\overline{\sigma}_i^T(I,a) = \frac{\sum_{t=1}^{T} h(t) \cdot \pi_i^{\sigma^t}(I) \cdot \sigma_i^t(I,a)}{\sum_{t=1}^{T} h(t)t \cdot \pi_i^{\sigma^t}(I)}$$

where $h(t)= t$

Deliverables: Implementation of CFR variants

- Establishing theoretical framework:

DCFR:



Discounting Factors:

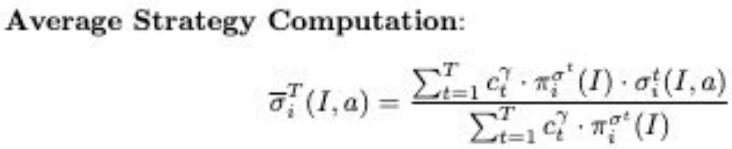
$$c_t^\alpha = \frac{t^\alpha}{t^\alpha + 1} \quad \text{(positive regrets)}$$

$$c_t^\beta = \frac{t^\beta}{t^\beta + 1} \quad \text{(negative regrets)}$$

$$c_t^\gamma = \frac{t^\gamma}{t^\gamma + 1} \quad \text{(average strategy)}$$

Cumulative Regret Update:

$$R_i^T(I,a) = \begin{cases} r_i^T(I,a) & \text{if } T = 1, \\ R_i^{T-1}(I,a) \cdot c_T^\alpha + r_i^T(I,a) & \text{if } R_i^{T-1}(I,a) > 0, \\ R_i^{T-1}(I,a) \cdot c_T^\beta + r_i^T(I,a) & \text{otherwise.} \end{cases}$$



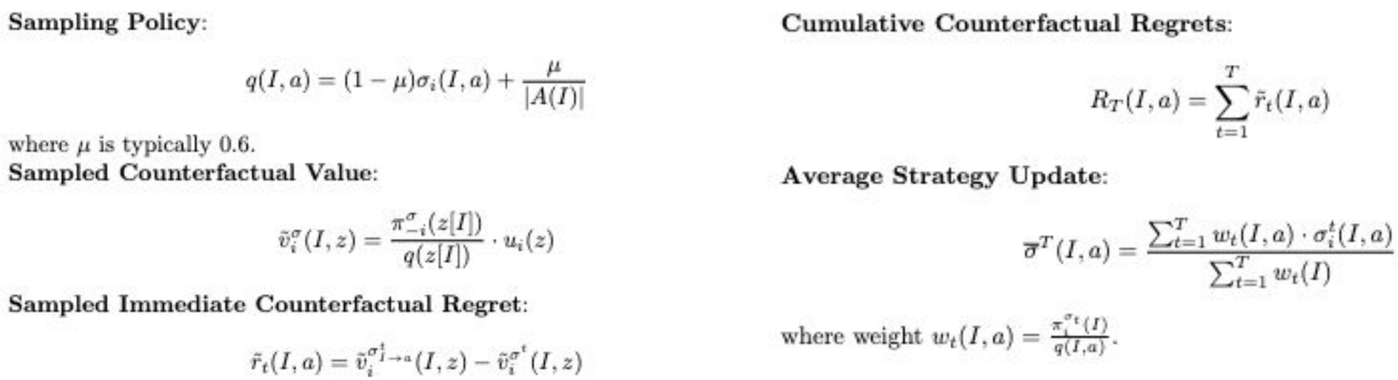
Average Strategy Computation:

$$\overline{\sigma}_i^T(I,a) = \frac{\sum_{t=1}^T c_t^\gamma \cdot \pi_i^{\sigma^t}(I) \cdot \sigma_i^t(I,a)}{\sum_{t=1}^T c_t^\gamma \cdot \pi_i^{\sigma^t}(I)}$$

Deliverables: Implementation of CFR variants

- Establishing theoretical framework:

MCCFR (OS-MCCFR):



Sampling Policy:

$$q(I,a) = (1-\mu)\sigma_i(I,a) + \frac{\mu}{|A(I)|}$$

where μ is typically 0.6.

Sampled Counterfactual Value:

$$\tilde{v}_i^{\sigma}(I,z) = \frac{\pi_{-i}^{\sigma}(z[I])}{q(z[I])} \cdot u_i(z)$$

Sampled Immediate Counterfactual Regret:

$$\tilde{r}_t(I,a) = \tilde{v}_i^{\sigma_{I\to a}^{t}}(I,z) - \tilde{v}_i^{\sigma^t}(I,z)$$

Cumulative Counterfactual Regrets:

$$R_T(I,a) = \sum_{t=1}^{T} \tilde{r}_t(I,a)$$

Average Strategy Update:

$$\overline{\sigma}^T(I,a) = \frac{\sum_{t=1}^{T} w_t(I,a) \cdot \sigma_i^t(I,a)}{\sum_{t=1}^{T} w_t(I)}$$

where weight $w_t(I,a) = \frac{\pi_i^{\sigma^t}(I)}{q(I,a)}$.

Deliverables: Implementation of CFR variants

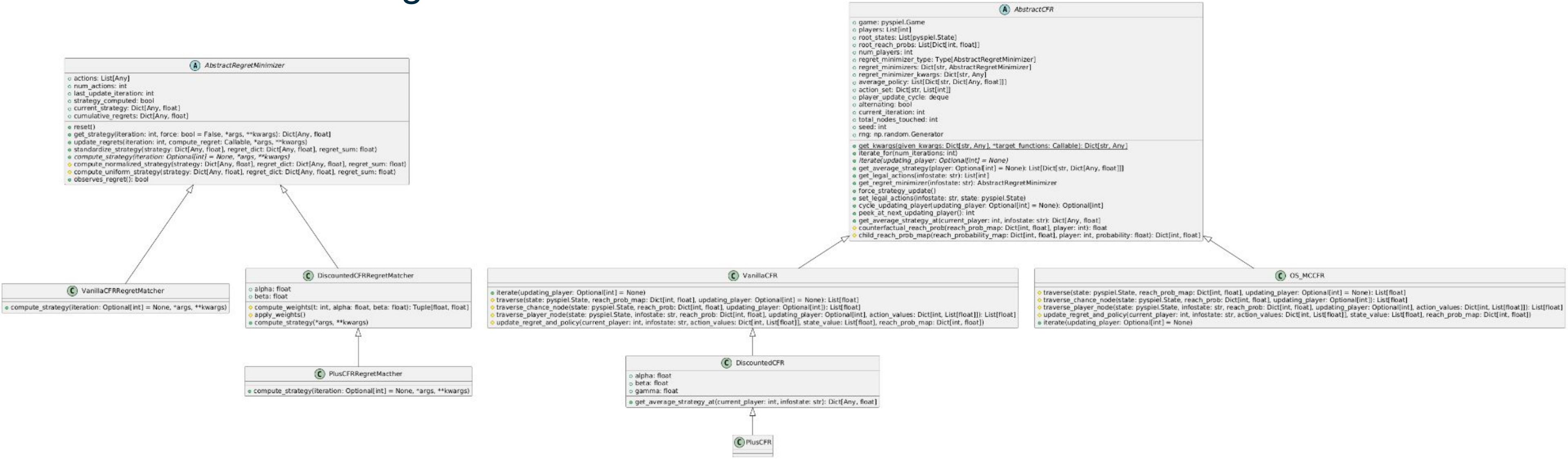
- Establishing technical framework:
  - Solution Design
  - Solution Implementation

- Solution Design:
  - **Abstract Base Classes:** Use inheritance to define common functionalities in abstract base classes and implement specific variants as subclasses.
  - **Clear Separation of Concerns:** Separate core CFR logic from regret minimization techniques and game-specific implementations. This promotes code reuse and simplifies the addition of new variants.
  - **Hierarchical Structure:**
    - **Regret Matching:** Base class defines the interface for core regret-matching operations. Subclasses like implement specific regret update and strategy calculation logic.
    - **CFR Algorithms:** Base class handles tree traversal and general CFR flow. Subclasses provide concrete implementations.

Deliverables: Implementation of CFR variants

- Establishing technical framework:
  - Solution Design
  - Solution Implementation

- Solution Design:



```
AbstractRegretMinimizer
○ actions: List[Any]
○ num_actions: int
○ last_update_iteration: int
○ strategy_computed: bool
○ current_strategy: Dict[Any, float]
○ cumulative_regrets: Dict[Any, float]
● reset()
● get_strategy(iteration: int, force: bool = False, *args, **kwargs): Dict[Any, float]
● update_regrets(iteration: int, compute_regret: Callable, *args, **kwargs)
● standardize_strategy(strategy: Dict[Any, float], regret_dict: Dict[Any, float], regret_sum: float)
● compute_strategy(iteration: Optional[int] = None, *args, **kwargs)
◇ compute_normalized_strategy(strategy: Dict[Any, float], regret_dict: Dict[Any, float], regret_sum: float)
◇ compute_uniform_strategy(strategy: Dict[Any, float], regret_dict: Dict[Any, float], regret_sum: float)
● observes_regret(): bool
```

```
VanillaCFRRegretMatcher
● compute_strategy(iteration: Optional[int] = None, *args, **kwargs)
```

```
DiscountedCFRRegretMatcher
○ alpha: float
○ beta: float
◇ compute_weights(t: int, alpha: float, beta: float): Tuple[float, float]
◇ apply_weights()
● compute_strategy(*args, **kwargs)
```

```
PlusCFRRegretMacther
● compute_strategy(iteration: Optional[int] = None, *args, **kwargs)
```

```
AbstractCFR
○ game: pyspiel.Game
○ players: List[int]
○ root_states: List[pyspiel.State]
○ root_reach_probs: List[Dict[int, float]]
○ num_players: int
○ regret_minimizer_type: Type[AbstractRegretMinimizer]
○ regret_minimizers: Dict[str, AbstractRegretMinimizer]
○ regret_minimizer_kwargs: Dict[str, Any]
○ average_policy: List[Dict[str, Dict[Any, float]]]
○ action_set: Dict[str, List[int]]
○ player_update_cycle: deque
○ alternating: bool
○ current_iteration: int
○ total_nodes_touched: int
○ seed: int
○ rng: np.random.Generator
● get_kwargs(given_kwargs: Dict[str, Any], *target_functions: Callable): Dict[str, Any]
● iterate_for(num_iterations: int)
● iterate(updating_player: Optional[int] = None)
● get_average_strategy(player: Optional[int] = None): List[Dict[str, Dict[Any, float]]]
● get_legal_actions(infostate: str): List[int]
● get_regret_minimizer(infostate: str): AbstractRegretMinimizer
● force_strategy_update()
● set_legal_actions(infostate: str, state: pyspiel.State)
● cycle_updating_player(updating_player: Optional[int] = None): Optional[int]
● peek_at_next_updating_player(): int
● get_average_strategy_at(current_player: int, infostate: str): Dict[Any, float]
◇ counterfactual_reach_prob(reach_prob_map: Dict[int, float], player: int): float
◇ child_reach_prob_map(reach_probability_map: Dict[int, float], player: int, probability: float): Dict[int, float]
```

```
VanillaCFR
● iterate(updating_player: Optional[int] = None)
◇ traverse(state: pyspiel.State, reach_prob_map: Dict[int, float], updating_player: Optional[int] = None): List[float]
◇ traverse_chance_node(state: pyspiel.State, reach_prob: Dict[int, float], updating_player: Optional[int]): List[float]
◇ traverse_player_node(state: pyspiel.State, infostate: str, reach_prob: Dict[int, float], updating_player: Optional[int], action_values: Dict[int, List[float]]): List[float]
◇ update_regret_and_policy(current_player: int, infostate: str, action_values: Dict[int, List[float]], state_value: List[float], reach_prob_map: Dict[int, float])
```

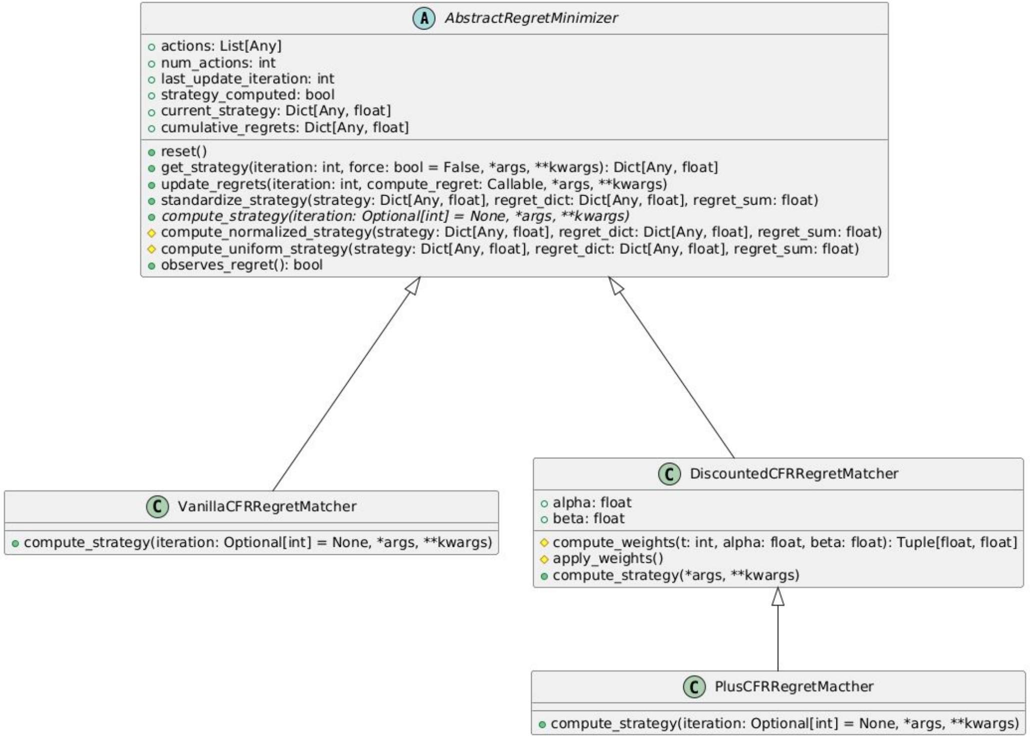
```
DiscountedCFR
○ alpha: float
○ beta: float
○ gamma: float
● get_average_strategy_at(current_player: int, infostate: str): Dict[Any, float]
```

```
PlusCFR
```

```
OS_MCCFR
◇ traverse(state: pyspiel.State, reach_prob_map: Dict[int, float], updating_player: Optional[int] = None): List[float]
◇ traverse_chance_node(state: pyspiel.State, reach_prob: Dict[int, float], updating_player: Optional[int]): List[float]
◇ traverse_player_node(state: pyspiel.State, infostate: str, reach_prob: Dict[int, float], updating_player: Optional[int], action_values: Dict[int, List[float]]): List[float]
◇ update_regret_and_policy(current_player: int, infostate: str, action_values: Dict[int, List[float]], state_value: List[float], reach_prob_map: Dict[int, float])
● iterate(updating_player: Optional[int] = None)
```

Deliverables: Implementation of CFR variants

- Solution Design:



A AbstractRegretMinimizer

○ actions: List[Any]
○ num_actions: int
○ last_update_iteration: int
○ strategy_computed: bool
○ current_strategy: Dict[Any, float]
○ cumulative_regrets: Dict[Any, float]

● reset()
● get_strategy(iteration: int, force: bool = False, *args, **kwargs): Dict[Any, float]
● update_regrets(iteration: int, compute_regret: Callable, *args, **kwargs)
● standardize_strategy(strategy: Dict[Any, float], regret_dict: Dict[Any, float], regret_sum: float)
● *compute_strategy(iteration: Optional[int] = None, *args, **kwargs)*
◇ compute_normalized_strategy(strategy: Dict[Any, float], regret_dict: Dict[Any, float], regret_sum: float)
◇ compute_uniform_strategy(strategy: Dict[Any, float], regret_dict: Dict[Any, float], regret_sum: float)
● observes_regret(): bool

C VanillaCFRRegretMatcher

● compute_strategy(iteration: Optional[int] = None, *args, **kwargs)

C DiscountedCFRRegretMatcher

○ alpha: float
○ beta: float

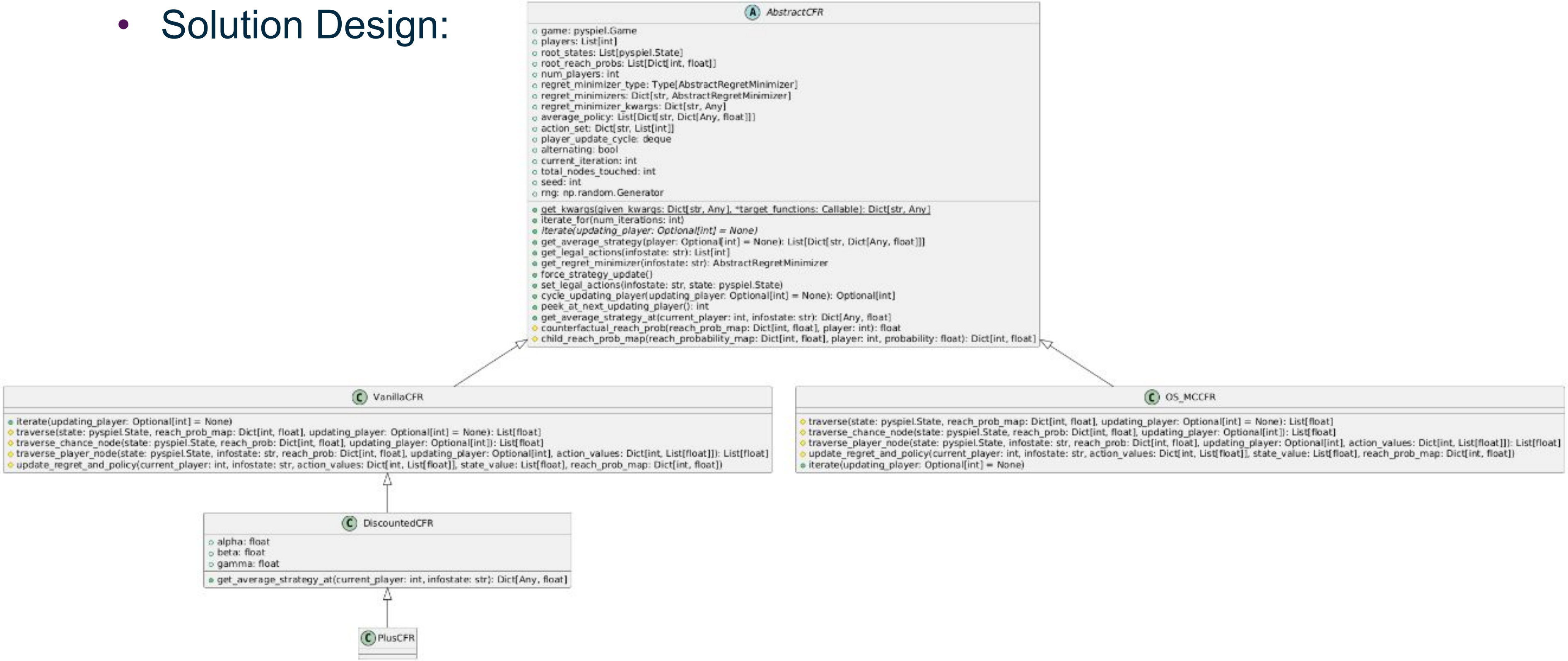
◇ compute_weights(t: int, alpha: float, beta: float): Tuple[float, float]
◇ apply_weights()
● compute_strategy(*args, **kwargs)

C PlusCFRRegretMacther

● compute_strategy(iteration: Optional[int] = None, *args, **kwargs)

Deliverables: Implementation of CFR variants

- Solution Design:



```
A AbstractCFR
○ game: pyspiel.Game
○ players: List[int]
○ root_states: List[pyspiel.State]
○ root_reach_probs: List[Dict[int, float]]
○ num_players: int
○ regret_minimizer_type: Type[AbstractRegretMinimizer]
○ regret_minimizers: Dict[str, AbstractRegretMinimizer]
○ regret_minimizer_kwargs: Dict[str, Any]
○ average_policy: List[Dict[str, Dict[Any, float]]]
○ action_set: Dict[str, List[int]]
○ player_update_cycle: deque
○ alternating: bool
○ current_iteration: int
○ total_nodes_touched: int
○ seed: int
○ rng: np.random.Generator
● get_kwargs(given_kwargs: Dict[str, Any], *target_functions: Callable): Dict[str, Any]
● iterate_for(num_iterations: int)
● iterate(updating_player: Optional[int] = None)
● get_average_strategy(player: Optional[int] = None): List[Dict[str, Dict[Any, float]]]
● get_legal_actions(infostate: str): List[int]
● get_regret_minimizer(infostate: str): AbstractRegretMinimizer
● force_strategy_update()
● set_legal_actions(infostate: str, state: pyspiel.State)
● cycle_updating_player(updating_player: Optional[int] = None): Optional[int]
● peek_at_next_updating_player(): int
● get_average_strategy_at(current_player: int, infostate: str): Dict[Any, float]
◇ counterfactual_reach_prob(reach_prob_map: Dict[int, float], player: int): float
◇ child_reach_prob_map(reach_probability_map: Dict[int, float], player: int, probability: float): Dict[int, float]
```

```
C VanillaCFR
● iterate(updating_player: Optional[int] = None)
◇ traverse(state: pyspiel.State, reach_prob_map: Dict[int, float], updating_player: Optional[int] = None): List[float]
◇ traverse_chance_node(state: pyspiel.State, reach_prob: Dict[int, float], updating_player: Optional[int]): List[float]
◇ traverse_player_node(state: pyspiel.State, infostate: str, reach_prob: Dict[int, float], updating_player: Optional[int], action_values: Dict[int, List[float]]): List[float]
◇ update_regret_and_policy(current_player: int, infostate: str, action_values: Dict[int, List[float]], state_value: List[float], reach_prob_map: Dict[int, float])
```

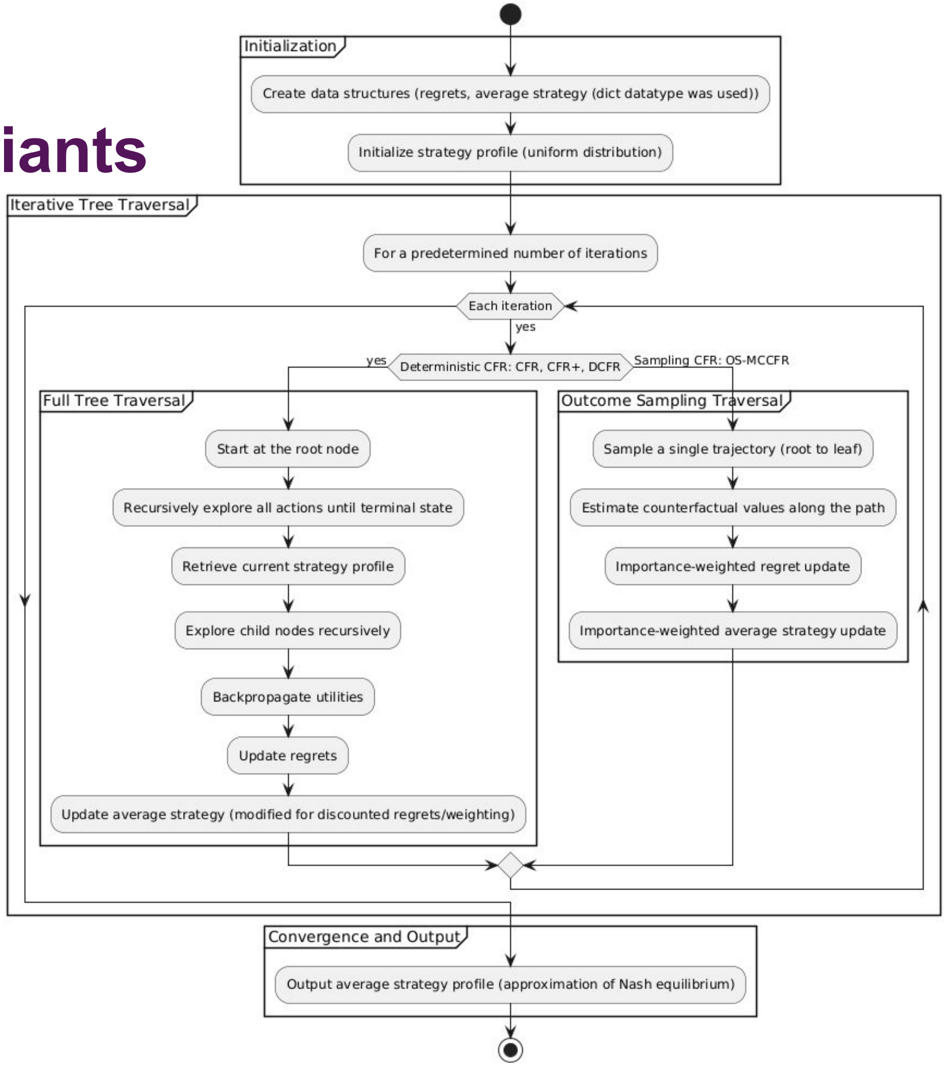
```
C OS_MCCFR
◇ traverse(state: pyspiel.State, reach_prob_map: Dict[int, float], updating_player: Optional[int] = None): List[float]
◇ traverse_chance_node(state: pyspiel.State, reach_prob: Dict[int, float], updating_player: Optional[int]): List[float]
◇ traverse_player_node(state: pyspiel.State, infostate: str, reach_prob: Dict[int, float], updating_player: Optional[int], action_values: Dict[int, List[float]]): List[float]
◇ update_regret_and_policy(current_player: int, infostate: str, action_values: Dict[int, List[float]], state_value: List[float], reach_prob_map: Dict[int, float])
● iterate(updating_player: Optional[int] = None)
```

```
C DiscountedCFR
○ alpha: float
○ beta: float
○ gamma: float
● get_average_strategy_at(current_player: int, infostate: str): Dict[Any, float]
```

```
C PlusCFR
```

Deliverables:
Implementation of CFR variants

- Solution Implementation:
  - The original CFR and its modifications are implemented to operate through a structured iterative process



Initialization

Create data structures (regrets, average strategy (dict datatype was used))

Initialize strategy profile (uniform distribution)

Iterative Tree Traversal

For a predetermined number of iterations

Each iteration

yes

yes

Deterministic CFR: CFR, CFR+, DCFR

Sampling CFR: OS-MCCFR

Full Tree Traversal

Start at the root node

Recursively explore all actions until terminal state

Retrieve current strategy profile

Explore child nodes recursively

Backpropagate utilities

Update regrets

Update average strategy (modified for discounted regrets/weighting)

Outcome Sampling Traversal

Sample a single trajectory (root to leaf)

Estimate counterfactual values along the path

Importance-weighted regret update

Importance-weighted average strategy update

Convergence and Output

Output average strategy profile (approximation of Nash equilibrium)

Deliverables: Implementation of CFR variants

- Implementation environment:
  - Python 3.11.5:
    - Rapid Prototyping
    - Readability
    - OpenSpiel framework
  - OpenSpiel (pyspiel library): Open-source framework to facilitate RL / AI research
    - Game Representation (game tree)
    - Utility functions (finding exploitability of the strategy)

Deliverables: Validation of implemented CFR variants

- Implementation convergence was validated using Kuhn poker, a well-understood game with a known optimal solution

- The project compared the average expected utility of the computed strategies to the known optimal average expected utility value

- Expected utility in this context refers to the average payoff a player can expect to receive when both players are using their respective strategies. The optimal expected utility is the expected payoff when both players are using optimal strategies
- Optimal expected utility in Kuhn poker is ± 1/18 ≈ ±0.0556

Deliverables: Validation of implemented CFR variants

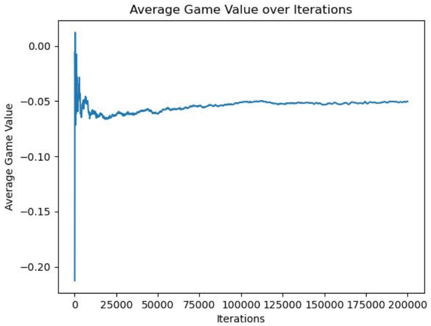


Fig 1. Convergence to optimal expected utility by implemented MCCFR

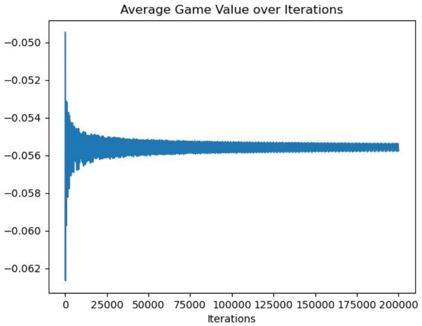


Fig 2. Convergence to optimal expected utility by implemented CFR

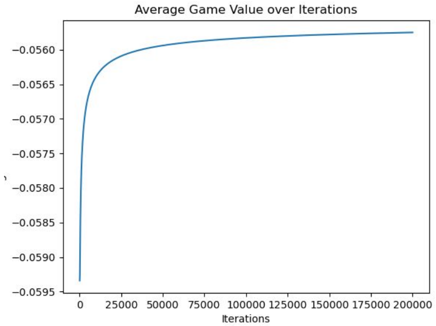


Fig 3. Convergence to optimal expected utility by implemented CFR+

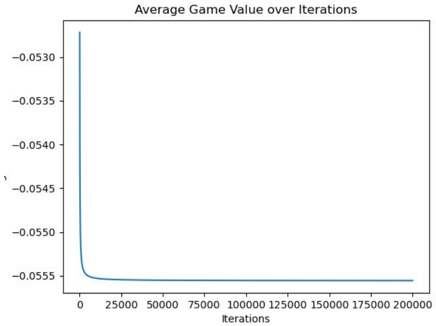


Fig 4. Convergence to optimal expected utility by implemented DCFR

Deliverables: Select and implement benchmark game environments

- Aims:
  - Building upon the established research by using recognised or forms of the recognised benchmarks
  - Varying complexity to allow scalability analysis as results are evaluated
  - Computationally feasible yet complex enough to differentiate algorithm performance
- The emphasis is on comparing the relative convergence speeds and scalability trends of CFR variants, rather than pushing the boundaries of solvable game sizes

Deliverables: Select and implement benchmark game environments

- Leduc Hold'em
  - A widely used benchmark in CFR research. A simple but representative imperfect-information poker game with 288 information sets. Provides a baseline for performance comparison
  - Six cards (two suits, three ranks) in the deck. Two betting rounds and a maximum of two raises per round

- Proposed variant: Short Deck Big Leduc Hold’em
  - Big Leduc Hold’em explands Leduc Hold’em by enlarging the deck size to twenty four cards (two suits, twelve ranks) and increasing number of maximum raises to six

Deliverables: Select and implement benchmark game environments

- Proposed variant: Short Deck Big Leduc Hold’em
  - Big Leduc Hold’em explands Leduc Hold’em by enlarging the deck size to twenty four cards (two suits, twelve ranks) and increasing number of maximum raises to six
  - However Big Leduc Hold’em contains 279312 information sets
  - A simplified variant is proposed to address the significant complexity jump between Leduc Hold'em and the original Big Leduc Hold'em with 12 cards in a deck (2 suits, 6 ranks)
  - This change reduces number of information sets to 31944

Deliverables: Comparative Analysis of CFR variants

- Aims:
  - To systematically compare the performance of different CFR variants:
    - Vanilla CFR (alternating and original non-alternating variants)
    - Discounted CFR (DCFR; original alternating and non-alternating variants)
    - CFR+ (with linear and quadratic weighted averaging)
    - Outcome Sampling Monte Carlo CFR (OS-MCCFR)
  - Two dimensions:
    - Convergence Analysis: How quickly does each algorithm converge towards a Nash equilibrium?
    - Scalability Analysis: How well does each algorithm's performance scale as the game complexity increases?

Deliverables: Comparative Analysis of CFR variants

- Aims:
  - To systematically compare the performance of different CFR variants:
    - Vanilla CFR (alternating and original non-alternating variants)
    - Discounted CFR (DCFR; original alternating and non-alternating variants)
    - CFR+ (with linear and quadratic weighted averaging)
    - Outcome Sampling Monte Carlo CFR (OS-MCCFR)
  - Two dimensions:
    - Convergence Analysis: How quickly does each algorithm converge towards a Nash equilibrium?
    - Scalability Analysis: How well does each algorithm's performance scale as the game complexity increases?

Deliverables: Comparative Analysis of CFR variants

- Methodology:
  - Exploitability as the primary metric:
    - Quantifies how much a strategy can be exploited by a theoretically optimal opponent
    - Lower exploitability indicates a closer approximation to a Nash equilibrium
  - Iteration-based evaluation
    - Algorithms were run for 1000 iterations in both benchmarks
    - Iteration counts are frequently used to measure CFR method performance in the literature. Furthermore it helped to manage computation time by measuring how much time iteration takes on average

Deliverables: Comparative Analysis of CFR variants

- CFR variants:
  - Vanilla CFR (CFR)
  - Vanilla CFR with Alternating Updates (CFR Alt.)
  - Discounted CFR (DCFR, $\alpha = 1.5$, $\beta = 0$, $\gamma = 2$)
  - Discounted CFR with Alternating Updates (DCFR Alt.) (original DCFR)
  - CFR+ with Linear Weighted Averaging (CFR+)
  - CFR+ with Quadratic Weighted Averaging (CFR+ (QWA))
  - Outcome Sampling Monte Carlo CFR (OS-MCCFR)

Deliverables: Comparative Analysis of CFR variants

- Results:

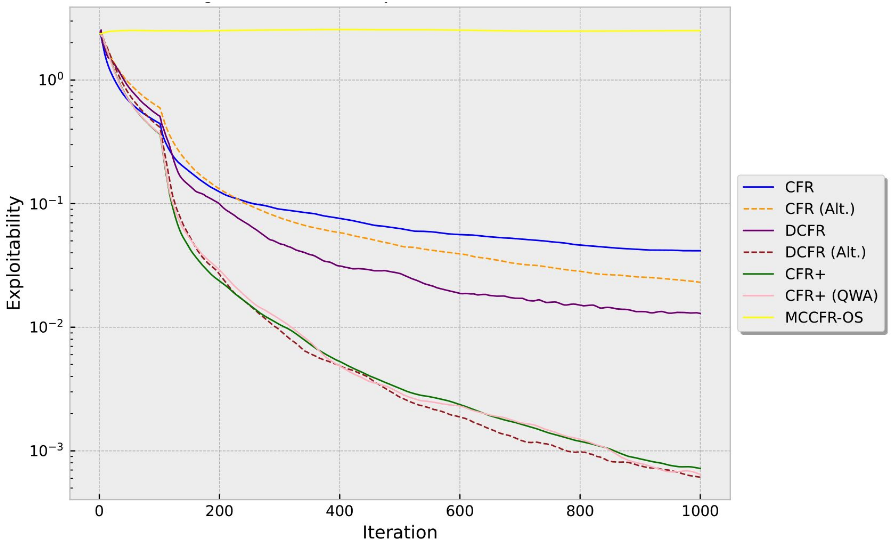


Fig 1. Convergence to Nash Equilibrium in Leduc Hold’em

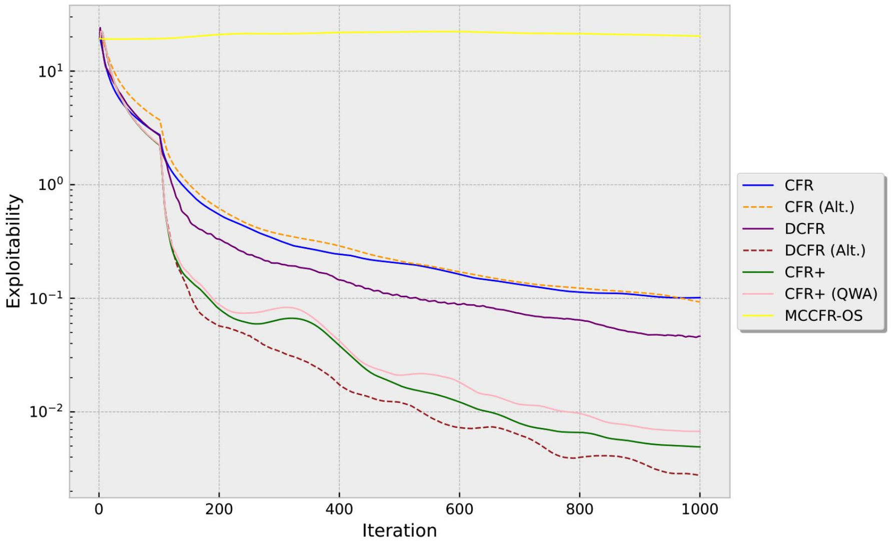
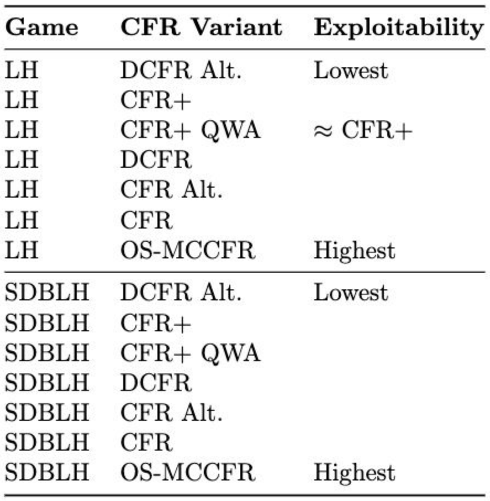


Fig 1. Convergence to Nash Equilibrium in Short Deck Big Leduc Hold’em

Deliverables: Comparative Analysis of CFR variants

- Convergence analysis:



Game	CFR Variant	Exploitability
LH	DCFR Alt.	Lowest
LH	CFR+	
LH	CFR+ QWA	≈ CFR+
LH	DCFR	
LH	CFR Alt.	
LH	CFR	
LH	OS-MCCFR	Highest
SDBLH	DCFR Alt.	Lowest
SDBLH	CFR+	
SDBLH	CFR+ QWA	
SDBLH	DCFR	
SDBLH	CFR Alt.	
SDBLH	CFR	
SDBLH	OS-MCCFR	Highest

Table 1: Basic Hierarchy (After 1000 Iterations)

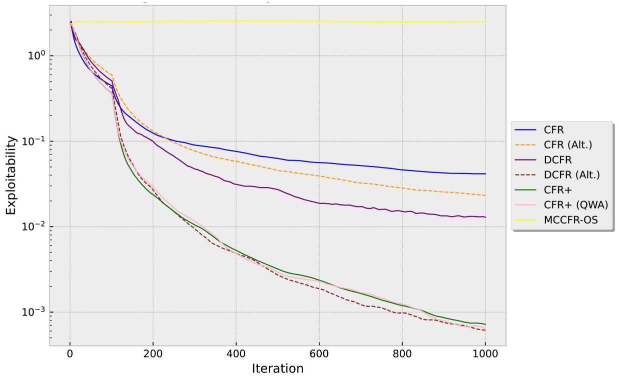


Fig 1. Convergence to Nash Equilibrium in Leduc Hold’em

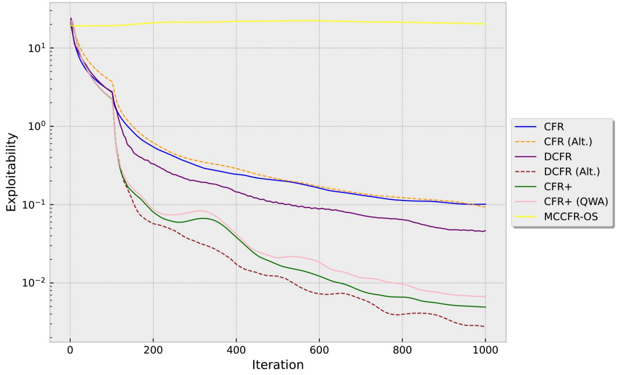


Fig 1. Convergence to Nash Equilibrium in Short Deck Big Leduc Hold’em

Deliverables: Comparative Analysis of CFR variants

- Convergence analysis (Leduc Hold’em):
  - DCFR (Alt.) Fastest: DCFR with alternating updates showed significantly faster convergence than all other tested methods, outperforming majority by multiple orders of magnitude.
  - DCFR vs. CFR+: DCFR (with alternating updates) and CFR+ show competitiveness, potentially contradicting existing claims that DCFR (α = 1.5, β = 0, γ = 2) consistently outperforms CFR+
  - Vanilla CFR: Vanilla CFR showed consistent but slow convergence. Alternating updates provided a slight improvement but less than an order of magnitude difference.
  - DCFR: Non-alternating DCFR converged at a rate comparable to standard CFR, showing less than an order of magnitude improvement.
  - CFR+ QWA vs. CFR+: CFR+ with quadratic weighted averaging reached lower exploitability than standard CFR+ after 1000 iterations.
  - OS-MCCFR: OS-MCCFR showed no signs of convergence compared to the other strategies within the 1000 iteration limit.

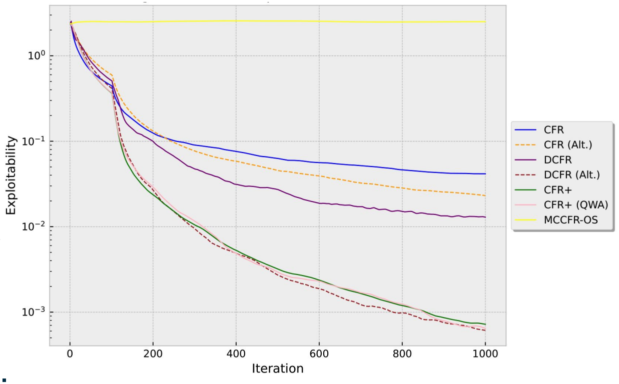


Fig 1. Convergence to Nash Equilibrium in Leduc Hold’em

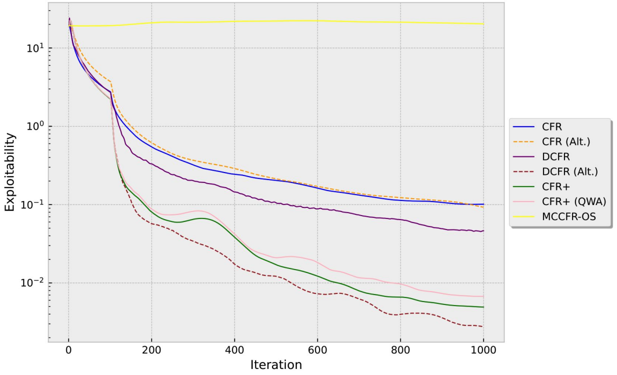


Fig 1. Convergence to Nash Equilibrium in Short Deck Big Leduc Hold’em

Deliverables: Comparative Analysis of CFR variants

- Convergence analysis
  (Short Deck Big Leduc Hold’em):
  - Higher Exploitability: All algorithms achieved higher exploitability in SDBLH compared to LH by approximately an order of magnitude, reflecting the increased game complexity.
  - DCFR (Alt.) Still Fastest: DCFR with alternating updates again showed the fastest convergence, though its lead over other algorithms was more pronounced in SDBLH than in LH.
  - CFR+ vs CFR+ (QWA): CFR+ significantly outperformed CFR+ with quadratic weighted averaging in SDBLH, contrasting their near-identical performance in LH.

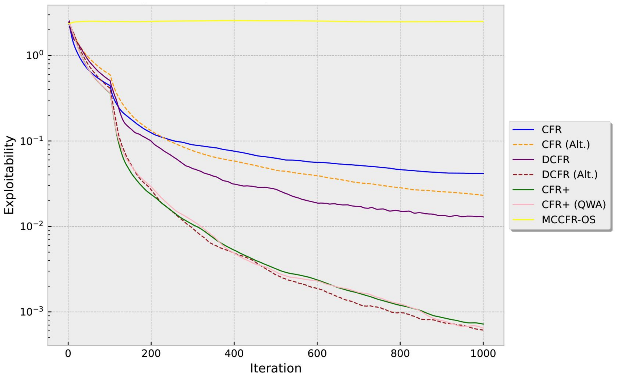


Fig 1. Convergence to Nash Equilibrium in Leduc Hold’em

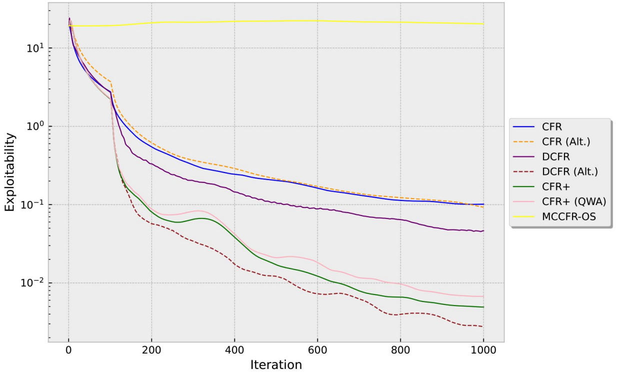


Fig 1. Convergence to Nash Equilibrium in Short Deck Big Leduc Hold’em

Deliverables: Comparative Analysis of CFR variants

- Scalability analysis:
  - Uniform Exploitability Increase: The uniform scaling of exploitability increase (approximately one order of magnitude) when moving from LH to SDBLH suggests that the increased complexity of SDBLH poses similar challenges to all tested algorithms.
  - The gap between standard CFR and CFR (Alt.) narrowed in SDBLH, suggesting that the benefits of alternating updates might diminish in larger games.
  - The performance difference between standard CFR+ and CFR+ (QWA) increased in SDBLH, with CFR+ outperforming the quadratic variant. This contrasts with expectations that quadratic weighted averaging might be more beneficial in larger games

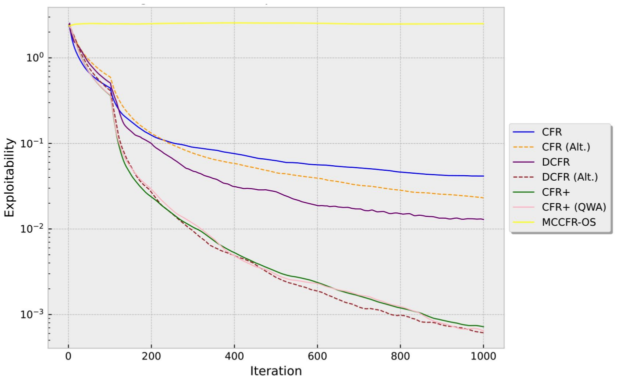


Fig 1. Convergence to Nash Equilibrium in Leduc Hold'em

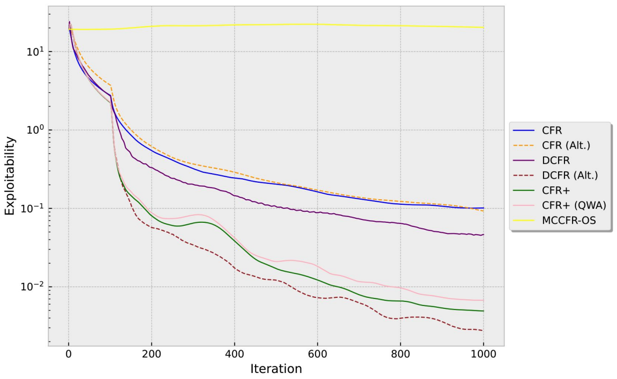


Fig 1. Convergence to Nash Equilibrium in Short Deck Big Leduc Hold'em

Solution demonstration

Conclusion
&
Invitation for questions