Regret minimization and applications to solving games

Tuomas Sandholm

Focus of this lecture

- We focus on Nash equilibrium in two-player zero-sum games
- But, these techniques apply to other problems as well, *e.g.:*
  - Best response computation
  - Quantal response equilibrium
    [Farina, Kroer, and Sandholm; "Online Convex Optimization for Sequential Decision Processes and Extensive-Form Games, AAAI'17]
  - Near-safe opponent exploitation
    [same as above]
  - Coarse-correlated equilibrium in multiplayer games
  - Playing better than a given strategy, but like it
    [Jacob, Wu, Farina, Lerer, Hu, Bakhtin, Andreas, Brown; Modeling Strong and Human-Like Gameplay with KL-Regularized Search. ICML'22]

Part One

Nash Equilibria in Normal-Form Games

Recap: Normal-Form Games

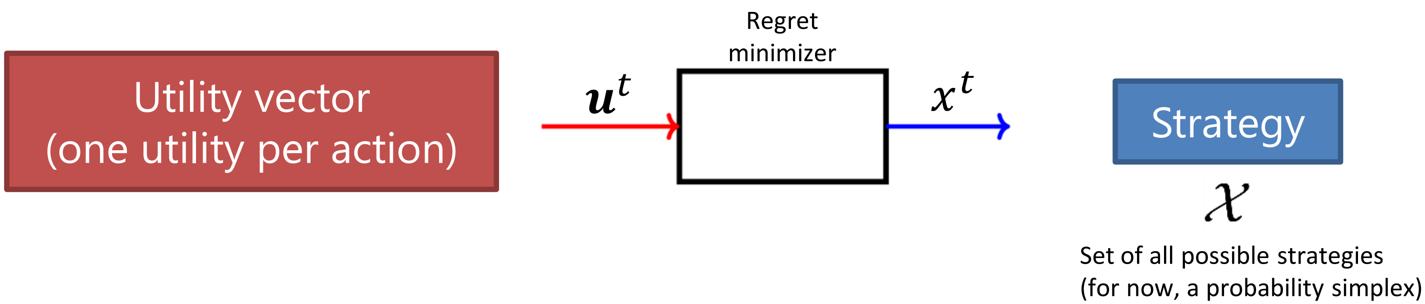
				
0.2		0	-1	+1
0.5		+1	0	-1
0.3		-1	+1	0

SIMULTANEOUS

(No turns)

Strategy for a player is just a probability distribution over actions

Regret Minimization



"How well do we do against ***best, fixed*** *strategy in hindsight?"*

$$R^T := \boxed{\max_{\hat{x} \in X} \left\{ \sum_{t=1}^{T} \langle u^t, \hat{x} \rangle \right\}} - \boxed{\sum_{t=1}^{T} \langle u^t, x^t \rangle}$$

Maximum utility that was achievable by the **best fixed** action in hindsight

Utility that was actually accumulated

Goal: have R^T grow sublinearly with respect to time T (e.g., $R^T \leq c\sqrt{T}$)

No assumption available on future utilities!

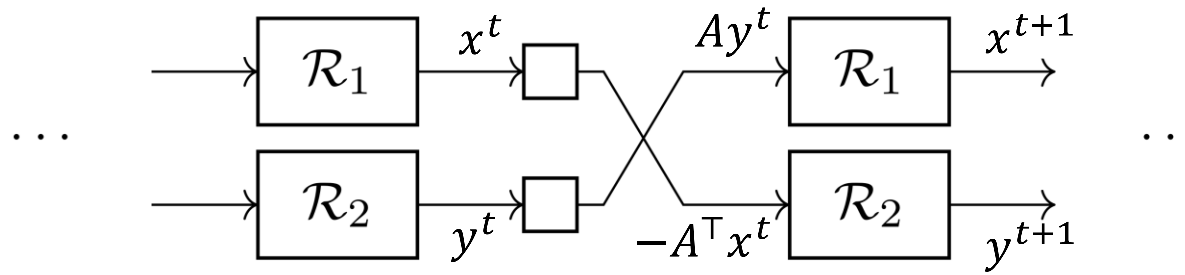
Must handle adversarial environments

Relationship with Nash Equilibrium

Nash equilibrium in a 2-player 0-sum normal-form game with payoff matrix A:

$$\max_{x \in \Delta^m} \min_{y \in \Delta^n} x^\top A y$$

IDEA: Self-play



$$R_1^T := \max_{\hat{x} \in \Delta^m} \left\{ \sum_{t=1}^{T} \langle Ay^t, \hat{x} \rangle \right\} - \cancel{\sum_{t=1}^{T} \langle Ay^t, x^t \rangle} \leq \sqrt{T}$$

$$R_2^T := \max_{\hat{y} \in \Delta^n} \left\{ \sum_{t=1}^{T} \langle -A^\top x^t, \hat{y} \rangle \right\} - \cancel{\sum_{t=1}^{T} \langle -A^\top x^t, y^t \rangle} \leq \sqrt{T}$$

Add these two lines and divide by T to get the average

$$\max_{\hat{x} \in \Delta^m} \left\{ \hat{x}^\top A \left(\frac{1}{T} \sum_{t=1}^{T} y^t \right) \right\} - \min_{\hat{y} \in \Delta^n} \left\{ \left(\frac{1}{T} \sum_{t=1}^{T} x^t \right)^\top A \hat{y} \right\} \leq \frac{2}{\sqrt{T}}$$

TAKEAWAY

The average strategies converge to a Nash equilibrium!

A Common Template for Regret Minimizers

- Given utility vectors $\mathrm{u}^1, \ldots, u^t$, we compute the empirical regrets up to time t of each action:

$$r^t[a] := \sum_{\tau=1}^{t} u^\tau[a] - \langle u^\tau, x^\tau \rangle$$

- Then, intuitively the next strategy x^{t+1} gives mass to actions somewhat proportionally to how much regret they have accumulated

A Common Template for Regret Minimizers

- Given utility vectors $u^1, \ldots, u^t$, we compute the empirical regrets up to time t of each action:

$$r^t[a] := \sum_{\tau=1}^{t} u^\tau[a] - \langle u^\tau, x^\tau \rangle$$

Hyperparameter ("learning rate")

-

Algorithm	Rule
Multiplicative weights update (MWU) (aka Hedge, aka Randomized Weighted Majority)	$x^{t+1}[a] = \dfrac{\exp\{\eta \cdot r^t[a]\}}{\sum_{a'} \exp\{\eta \cdot r^t[a']\}}$
Regret matching (RM)	$x^{t+1}[a] = \dfrac{\max\{0, r^t[a]\}}{\sum_{a'} \max\{0, r^t[a']\}}$

Note: MWU is a particular instance of a very general algorithm called "Online mirror descent", which can be applied to all convex strategy sets and guarantees sublinear regret

A Common Template for Regret Minimizers

Empirical regret: $r^t[a] \coloneqq \sum_{\tau=1}^{t} u^\tau[a] - \langle u^\tau, x^\tau \rangle$

Simple modification:

$$r_+^t[a] \coloneqq \max\{0, r_+^{t-1}[a] + u^t[a] - \langle u^t, x^t \rangle\}$$

Algorithm	Rule
Multiplicative weights update (MWU) (aka Hedge, aka Randomized Weighted Majority)	$x^{t+1}[a] = \frac{\exp\{\eta \cdot r^t[a]\}}{\sum_{a'} \exp\{\eta \cdot r^t[a']\}}$
Regret matching (RM)	$x^{t+1}[a] = \frac{\max\{0, r^t[a]\}}{\sum_{a'} \max\{0, r^t[a']\}}$
Regret matching plus (RM+)	$x^{t+1}[a] = \frac{\max\{0, r_+^t[a]\}}{\sum_{a'} \max\{0, r_+^t[a']\}}$

State-of-the-art variant in practice: **Discounted RM (DRM)**

- Linear RM (LRM)
  - Weight iteration t by t (in regrets and averaging)
  - RM+ floors regrets at 0. Can we combine this with linear RM? Theory: Yes. Practice: No! Does very poorly.
- But less-aggressive combinations do well: **Discounted RM**
  - On each iteration, multiply positive regrets by t^{α} / t^{α}+1
  - On each iteration, multiply negative regrets by t^{β} / t^{β}+1
  - Weight contributions toward average strategy by $(t / (t+1))^{\gamma}$
  - Worst-case convergence bound only a small constant worse than that of RM
  - For α = 1.5, β = 0, γ = 2, consistently outperforms RM+ in practice

[Brown & Sandholm, Solving Imperfect-Information Games via Discounted Regret Minimization, AAAI'19]

A Common Template for Regret Minimizers

All of these algorithms guarantee that after seeing any number T of utilities $u^1, \dots, u^T$, the regret cumulated by the algorithm satisfies

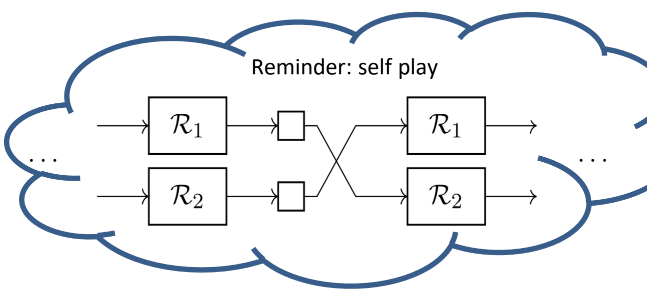
$$R^T \leq c \sqrt{\sum_{t=1}^{T} \|u^t\|_2^2}$$

Constant that depends on number of actions

Remember:
This holds without any assumption about the way the utilities are selected by the environment!

So, assuming that the utility vectors have bounded norms $\|u^t\| \leq B$ (this is always the case when playing finite games), then $R^T \leq cB\sqrt{T}$

Consequence: when using these algorithms in self-play in 2-player 0-sum games, the average strategy converges to a Nash equilibrium at a rate of $\frac{\sqrt{T}}{T} = \frac{1}{\sqrt{T}}$



What Regret Minimizers are Used in Practice?

Multiplicative Weights Update (MWU)

- ✓ Special case of OMD, that works for general convex sets
- ✓ Widely used & understood
- ✗ Slow in practice for games
- ✗ Hyperparameters (stepsize)

- ✓ Can incorporate optimism about future losses to converge faster in 2-player 0-sum games

Regret Matching (RM) & Regret Matching+ (RM+)

- ✗ Only for **simplex** domains
- ✗ Not as well studied
- ✓ Tuned for game solving
- ✓ No hyperparameters
- ✓ Incredibly effective

Modern variants of this, such as DCFR, are the standard in extensive-form game solving!

- ? Unknown… Until recently
- ✓

[Farina et al., Faster Game Solving via Predictive Blackwell Approachability: Connecting Regret Matching and Mirror Descent, AAAI 2021]

Optimistic regret minimizers

Algorithm	Standard (non-optimistic) rule	Optimistitic (aka Predictive) rule
MWU	$x^{t+1}[a] = \dfrac{\exp\{\eta \cdot r^t[a]\}}{\sum_{a'} \exp\{\eta \cdot r^t[a']\}}$	$x^{t+1}[a] = \dfrac{\exp\{\eta \cdot (r^t[a] + u^t[a] - \langle u^t, x^t \rangle)\}}{\sum_{a'} \exp\{\eta \cdot (r^t[a'] + u^t[a'] - \langle u^t, x^t \rangle)\}}$
RM	$x^{t+1}[a] = \dfrac{\max\{0, r^t[a]\}}{\sum_{a'} \max\{0, r^t[a']\}}$	$x^{t+1}[a] = \dfrac{\max\{0, r^t[a] + u^t[a] - \langle u^t, x^t \rangle\}}{\sum_{a'} \max\{0, r^t[a'] + u^t[a'] - \langle u^t, x^t \rangle\}}$
RM+	$x^{t+1}[a] = \dfrac{\max\{0, r_+^t[a]\}}{\sum_{a'} \max\{0, r_+^t[a']\}}$	$x^{t+1}[a] = \dfrac{\max\{0, r_+^t[a] + u^t[a] - \langle u^t, x^t \rangle\}}{\sum_{a'} \max\{0, r_+^t[a'] + u^t[a] - \langle u^t, x^t \rangle\}}$

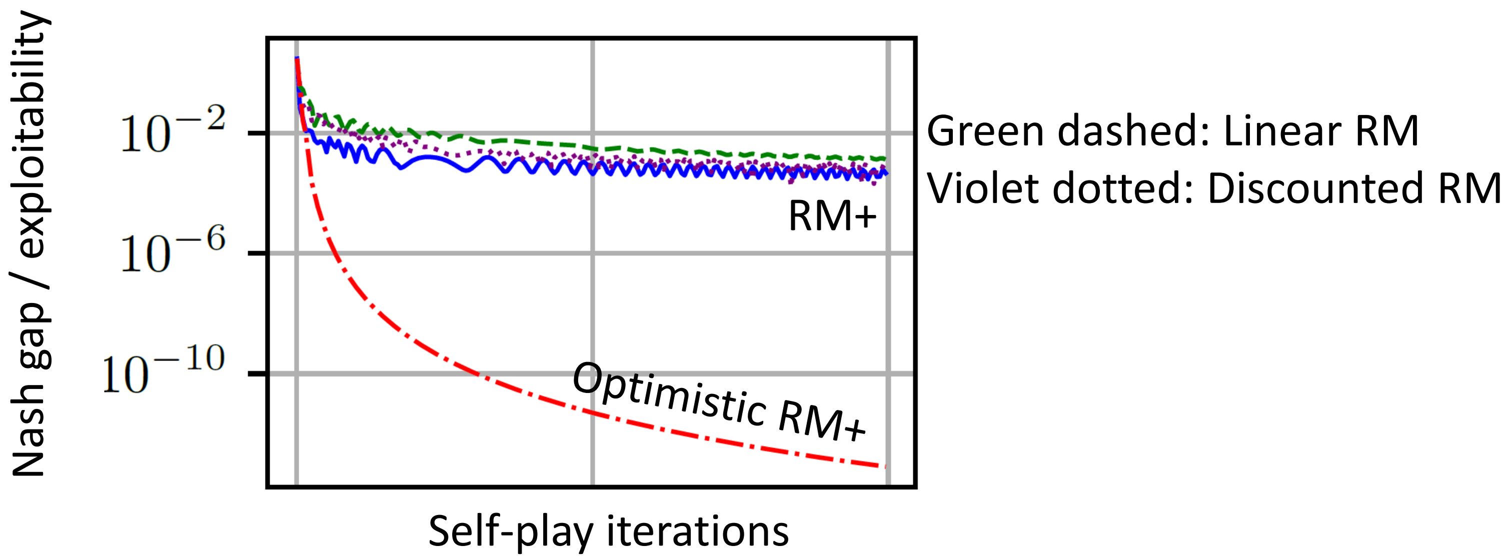
Typically, one-line change in implementation

All of these algorithms guarantee that after seeing any number T of utilities $u^1, \dots, u^T$, the regret cumulated by the algorithm satisfies

$$R^T \le c \sqrt{\sum_{t=2}^{T} \|u^t - u^{t-1}\|_2^2 + (\langle u^t, x^t \rangle - \langle u^{t-1}, x^{t-1} \rangle)^2}$$

Remember:
This holds without any assumption about the way the utilities are selected by the environment!

Takeaway message: still $\approx \sqrt{T}$ regret, but much smaller when there is little change to the utilities over time



(RM was omitted as it is typically much slower than RM+)

[Farina, Kroer, and Sandholm; Faster Game Solving via Predictive Blackwell Approachability: Connecting Regret Matching and Mirror Descent, AAAI’21]

- In general, Discounted RM and Optimistic RM+ are the fastest in practice
  - For some games, like poker, Discounted RM is empirically consistently faster than Optimistic RM+
  - For many other games, Optimistic RM+ is significantly faster

[Farina, Kroer, and Sandholm; Faster Game Solving via Predictive Blackwell Approachability: Connecting Regret Matching and Mirror Descent, AAAI'21]

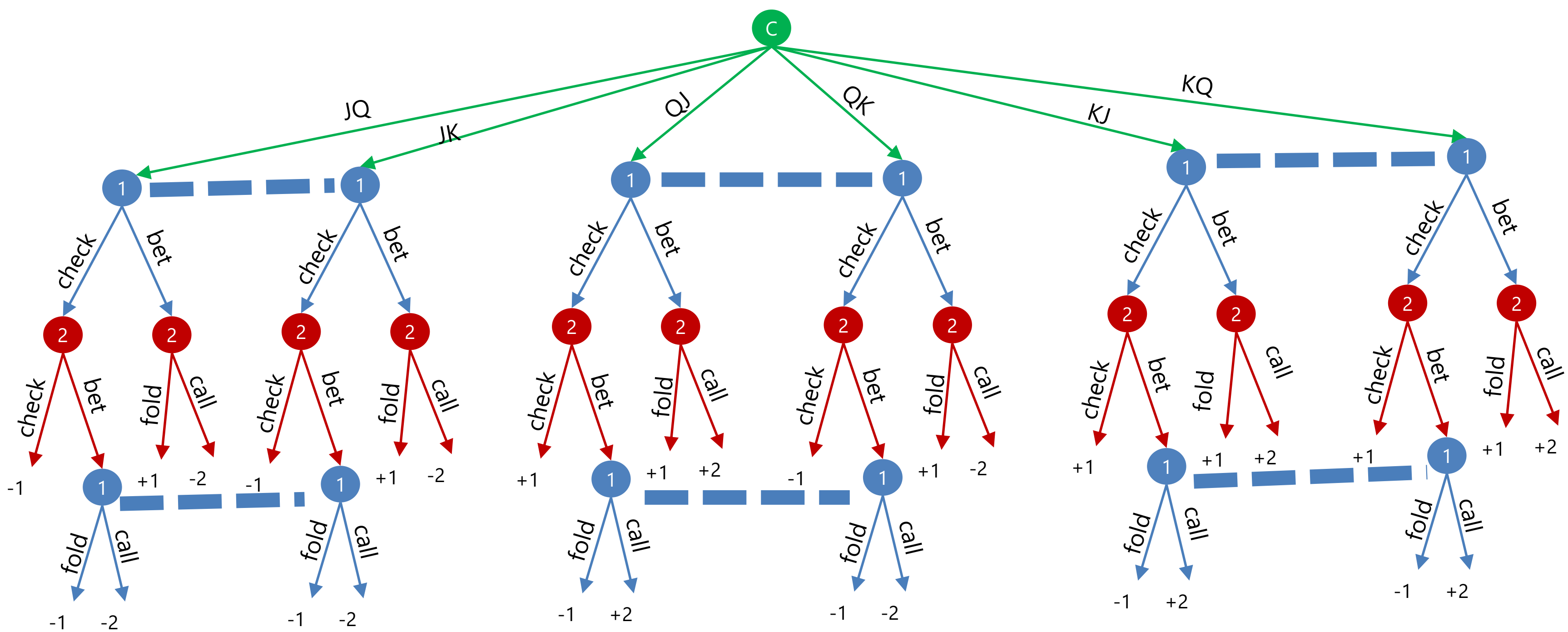
Part Two

From Normal-Form to **Extensive-Form**

(Nash Equilibria in Extensive-Form Games)

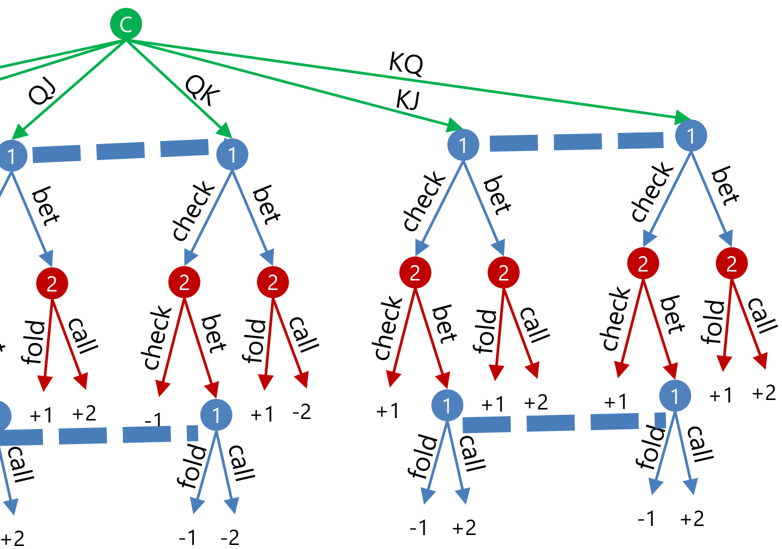
From Normal-Form to Extensive-Form

- Can capture sequential and simultaneous moves
- Private information
- We assume perfect recall: no player forgets what the player knew earlier



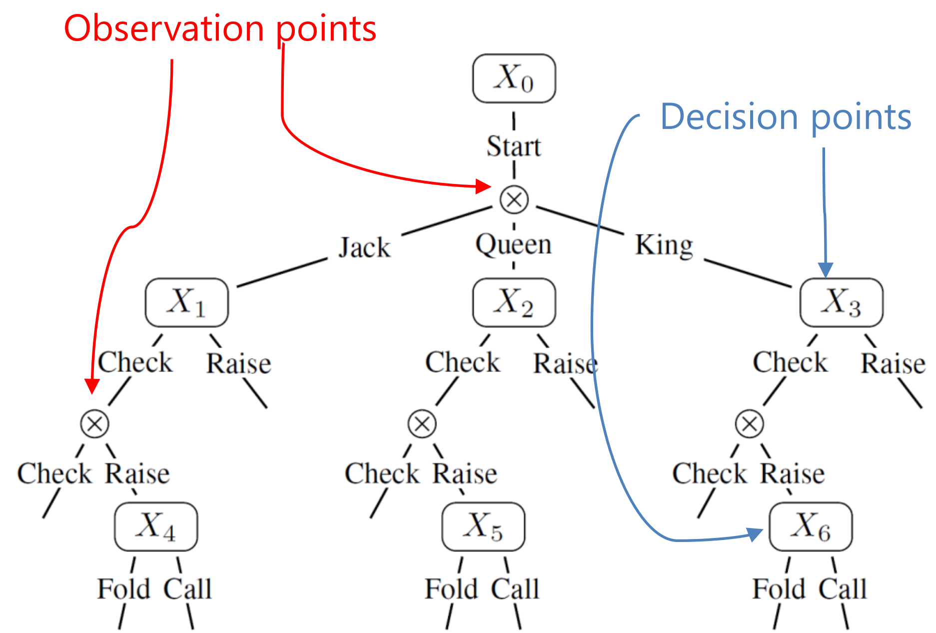
Information sets for red player (Player 2) are not shown

Two Representations



Game tree

Each node belongs to a specific player or chance

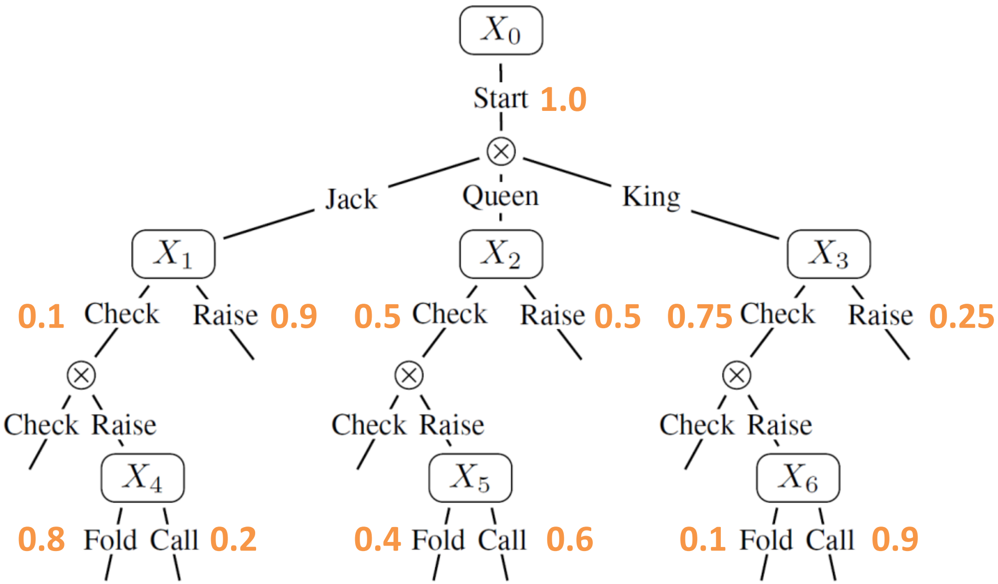


Tree-Form (Sequential) Decision Problem

Represents the game from viewpoint of one player

This is the representation in regret minimization

Strategies in Extensive-Form Games



``Behavioral strategies"

First attempt:

Assign local probabilities at each decision point

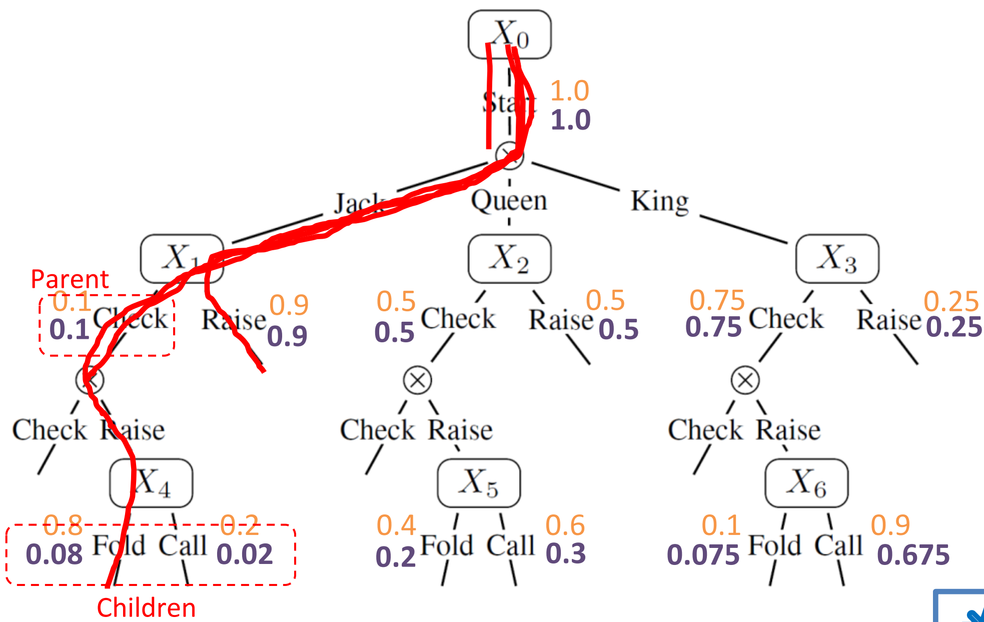
✓ Set of strategies is convex

✗ Expected utility of game is **not** bilinear

Reason: prob. of reaching a terminal state is product of variables

Products = non-convexity 😩

Strategies in Extensive-Form Games



``Sequence-form strategies"

Second attempt:

Store probabilities for whole sequences of actions

✔ Set of strategies is convex

✔ Expected utility of game is bilinear

Consistency constraints

1. Entries all non-negative
2. Root sequence has probability 1.0
3. Probability mass conservation

[Romanovskii, Reduction of a game with complete memory to a matrix game, 1961]
[Koller et al., Fast algorithms for finding randomized strategies in game trees, STOC 1994]

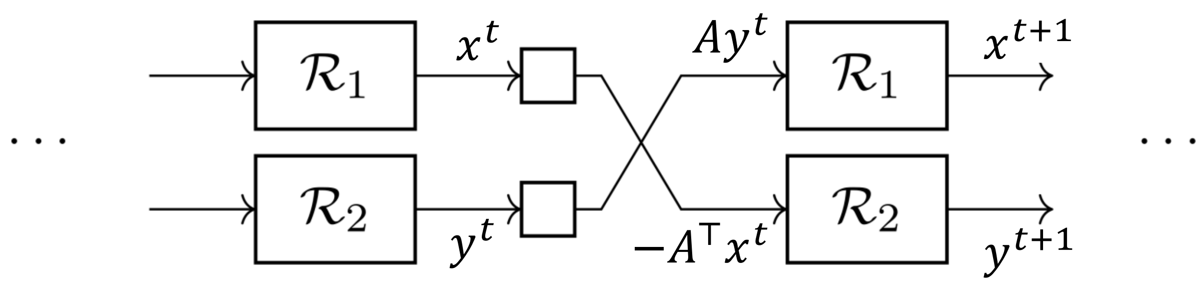
Sequence-Form Strategies

Consequence: a lot of results carry over from normal form games when using sequence-form strategies!

- Nash equilibrium is a bilinear saddle point problem

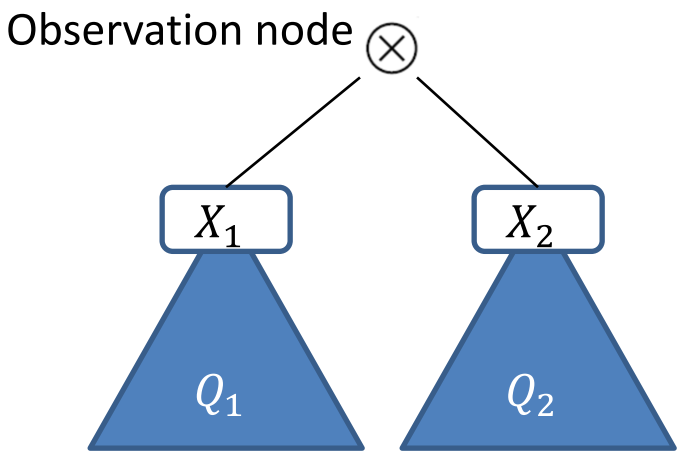
$$\min_{x \in Q_1} \max_{y \in Q_2} x^\top A y$$

- As long as we can construct regret minimizers for the sets of sequence-form strategies, we can use them to converge to Nash equilibrium in self play



Regret Minimizers for Sequence-Form Strategy Spaces

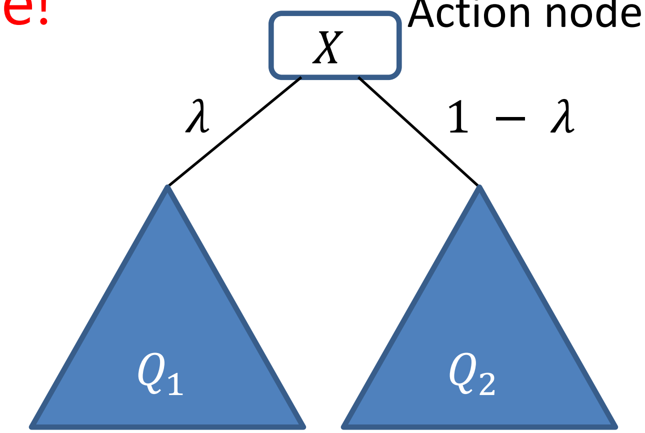
IDEA: Sequence-form strategy spaces have a strong combinatorial structure!



Any (q_1, q_2) is a valid s.f. strategy

$$Q = Q_1 \times Q_2$$

Cartesian Products



Any $(\lambda, 1-\lambda, \lambda q_1, (1-\lambda) q_2)$ is a valid s.f. strategy

$$Q = \operatorname{conv}\left(\begin{pmatrix} 1 \\ 0 \\ Q_1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ Q_2 \end{pmatrix} \right)$$

Convex Hulls

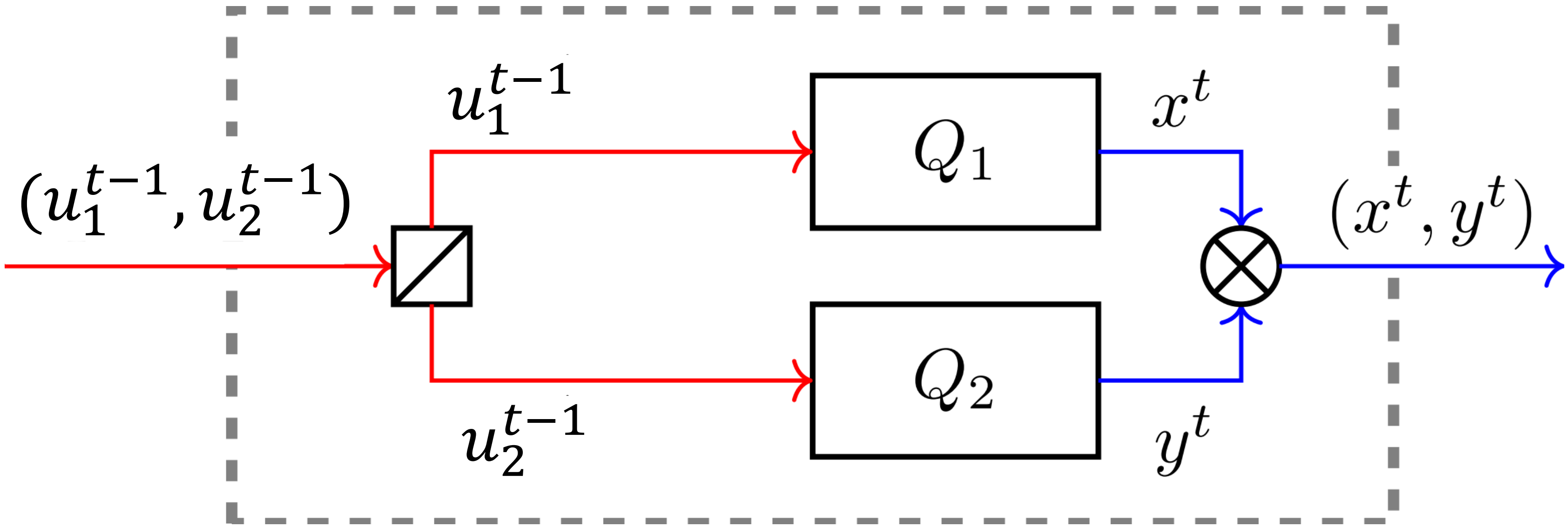
We can leverage this combinatorial structure to construct compositionally a regret minimizer for a player's sequence-form strategy set using any regret minimizers for simplices

☆

In other words: we can "reduce" regret minimization for extensive-form games to regret minimization for normal-form games

[Farina, Kroer & Sandholm, Regret Circuits: Composability of Regret Minimizers, ICML-19]

Regret Circuits: Cartesian Product

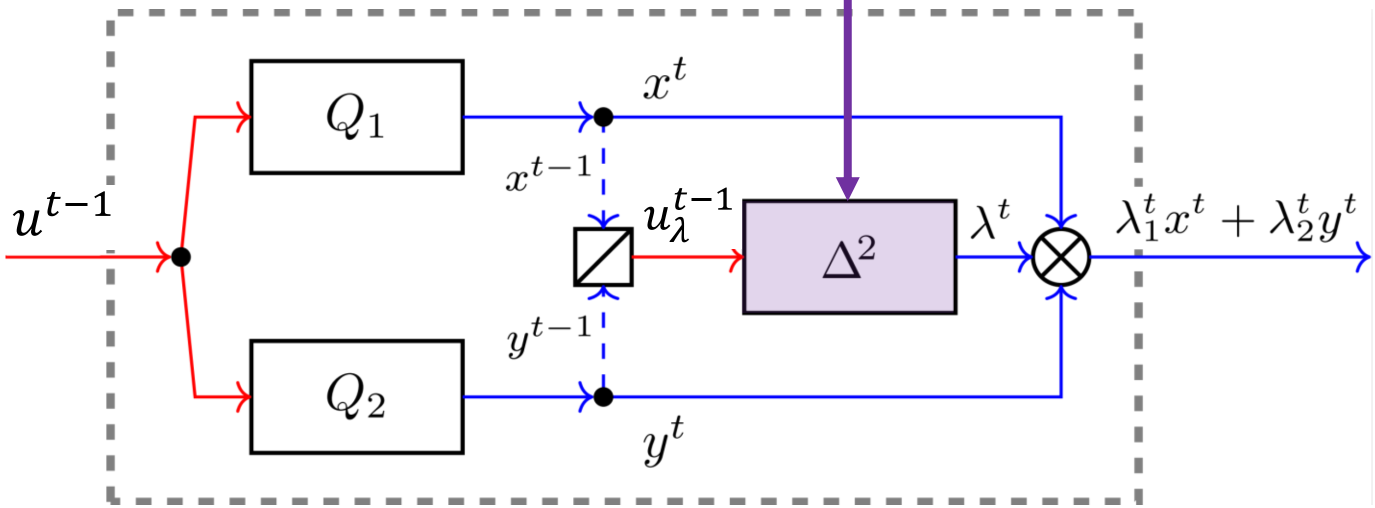


[Farina, Kroer & Sandholm, Regret Circuits: Composability of Regret Minimizers, ICML-19]

Regret Circuits: Convex Hull

IDEA

Extra regret minimizer decides how to mix the decisions on X and Y



Any will work!
RM, RM+, DRM, Hedge, OMD, FTRL, ….

[Farina, Kroer & Sandholm, Regret Circuits: Composability of Regret Minimizers, ICML-19]

A generalized Counterfactual Regret Minimization (CFR) algorithm

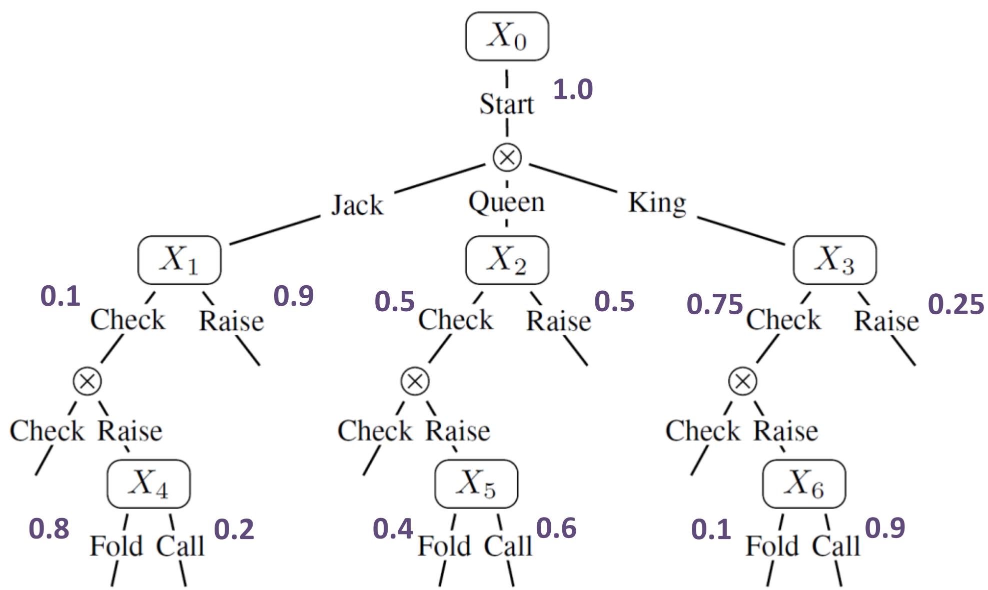
- So, by composing the constructions for Cartesian product and convex hull, we obtain a regret minimizer for extensive-form domains
- **IDEA:** decomposes and bounds regret locally at each decision point in the game
- This regret minimizer is called (generalized) CFR

This, with extensions discussed later, is the practical state-of-the-art in extensive-form game solving

- CFR: [Zinkevich et al., Regret minimization in games with incomplete information, NeurIPS-07]
- Generalized CFR: [Farina, Kroer & Sandholm, Regret Circuits: Composability of Regret Minimizers, ICML-19]

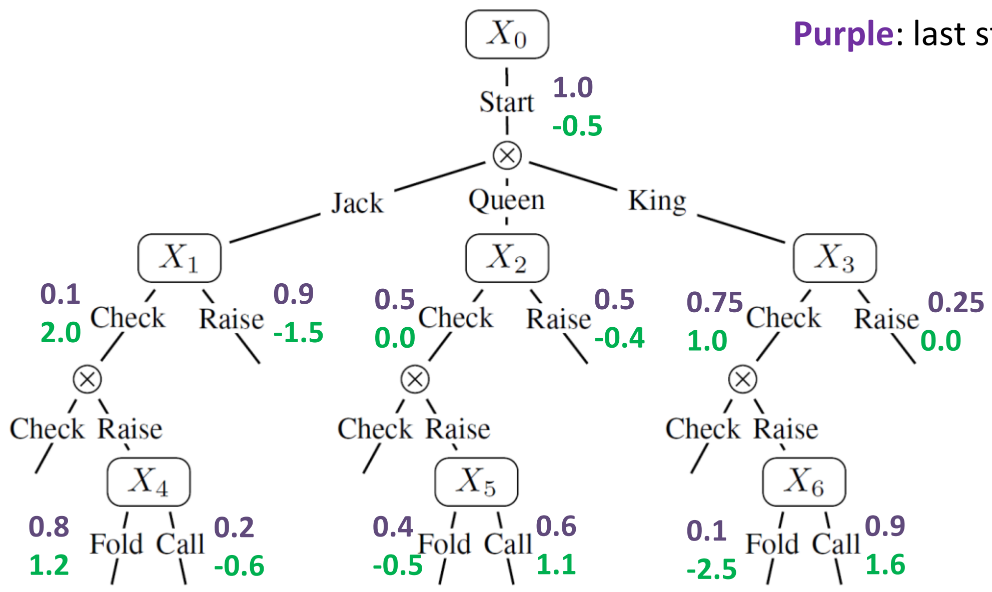
CFR Viz: Regret Minimization Algorithm

- How does CFR end up updating strategies?
- Suppose the following local strategies were output by the regret minimizers at each decision point:



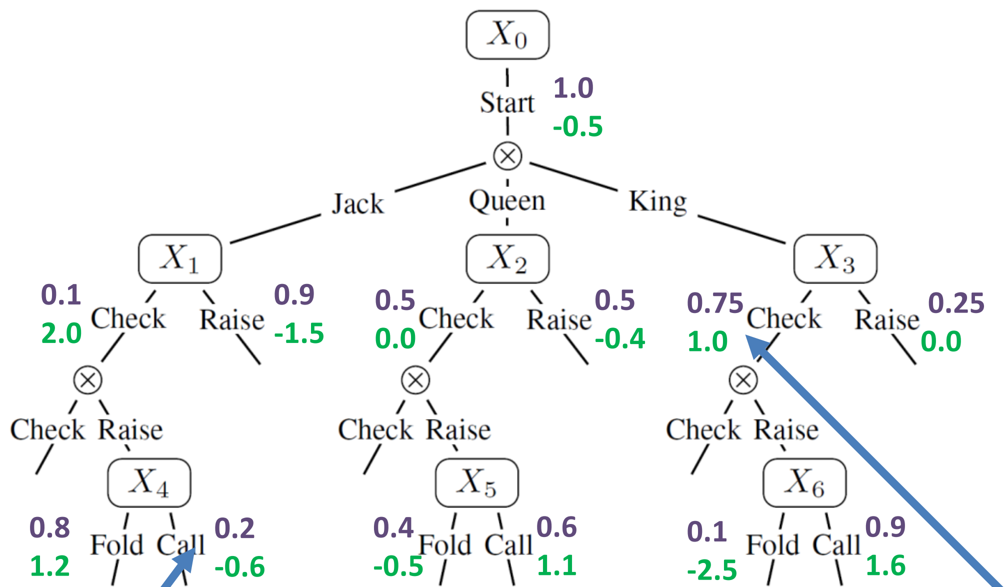
CFR Viz: Regret Minimization Algorithm

- An example utility vector u^t shown in green is now received by the CFR regret minimizer
  - Remember: in regret minimization we make no assumption as to how the environment picked the utility vector. So, the green utilities may not actually be “real” payoffs in Kuhn
  - In the special case of playing against an opponent to compute Nash, the green number = $\sum_{\text{subsequent \textbf{game tree} leaves in which this player can't move again because game ends}}$ leaf payoff * reach probability of opponent from root to that leaf * reach probability of chance from root to that leaf



Purple: last strategy output by each local regret minimizer

CFR Viz: Regret Minimization Algorithm



Step 1: Compute the **counterfactual utilities for each action**

At leaves:

counterfactual utility = utility at leaf

Elsewhere:

counterfactual utility =

utility at sequence

\+

expected utility in the subtree of the decision problem (not game tree) under that sequence, weighting actions according to latest strategy

Example:

Counterfactual utility = -0.6

Example:

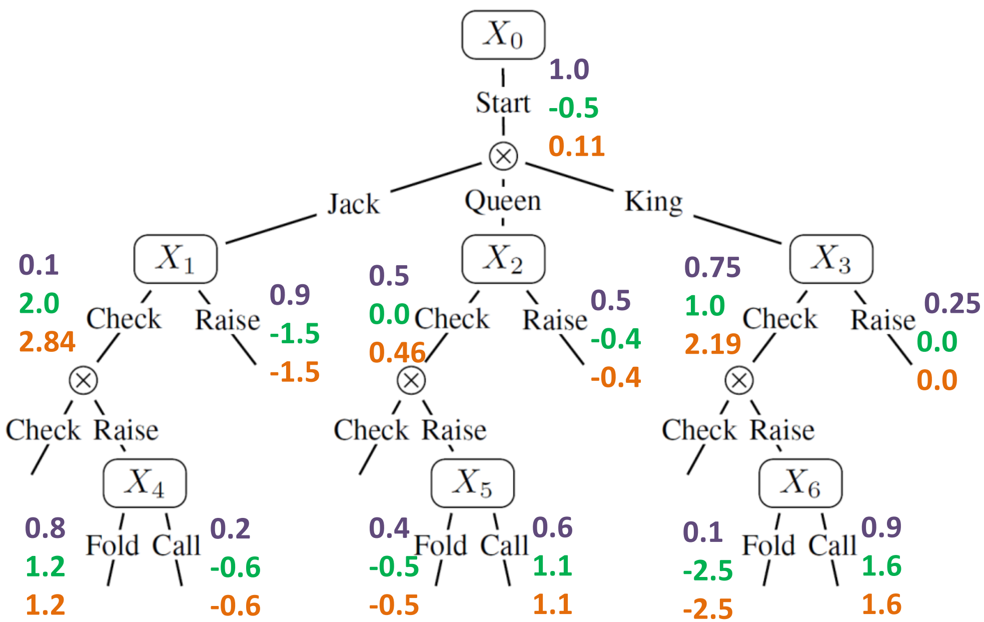
Counterfactual utility

= 1.0 + (0.1 x -2.5 + 0.9 x 1.6) = 2.19

Purple: last strategy output by each local regret minimizer

Green: utility vector given by environment

CFR Viz: Regret Minimization Algorithm



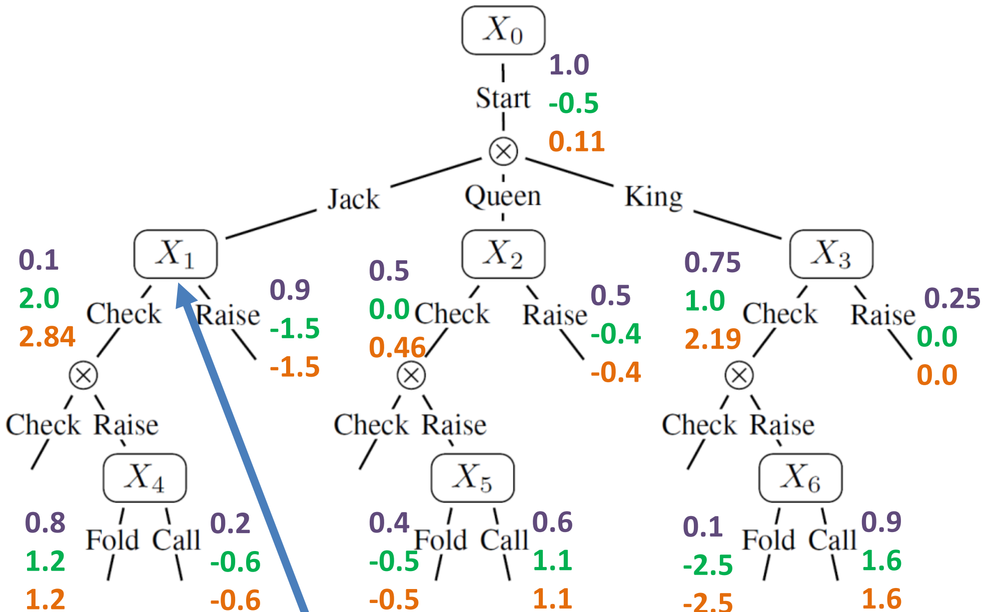
Step 1: Compute the **counterfactual utilities for each action**

At leaves:
counterfactual utility = utility at leaf

Elsewhere:
counterfactual utility =
utility at sequence
+
expected utility in the subtree of the decision problem (not game tree) under that sequence, weighting actions according to latest strategy

Purple: last strategy output by each local regret minimizer
Green: utility vector given by environment
Orange: counterfactual utilities

CFR Viz: Regret Minimization Algorithm



Example:

Feed utilities (2.84, -1.5) to the regret minimizer that selects the strategy for this decision point

Step 2: Feed the counterfactual utilities to the regret minimizers at each decision point

Note: Steps 1 & 2 can be done in a single bottom-up pass

Remember: this will work *no matter the choice of regret minimizers* (MWU, RM, RM+, Discounted RM, Optimistic RM+, etc). **We can also use different regret minimizers at different nodes, unlike the original CFR paper, which used RM**

Purple: last strategy output by each local regret minimizer
Green: utility vector given by environment
Orange: counterfactual utilities

CFR Guarantees

- **Theorem**: the regret cumulated by CFR can be bounded as

$$R_{CFR} \leq \sum_{\underbrace{j \in J}_{\text{Decision points}}} \max\{0, \underbrace{R_j}_{\text{Regret cumulated by local regret minimizer for decision point j}}\}$$

- *Consequence*: if the local regret minimizers guarantee sublinear regret, then CFR cumulates sublinear regret

[Farina, Kroer & Sandholm, Regret Circuits: Composability of Regret Minimizers, ICML 2019]

Why is CFR Superior in Practice?

… to second-order methods (which can offer convergence rate $1/e^T$)?

- Does not require solving large linear systems
- Second-order methods (interior point, …) don't fit in memory for large games

… to general-purpose regret minimizers (FTRL & OMD)?

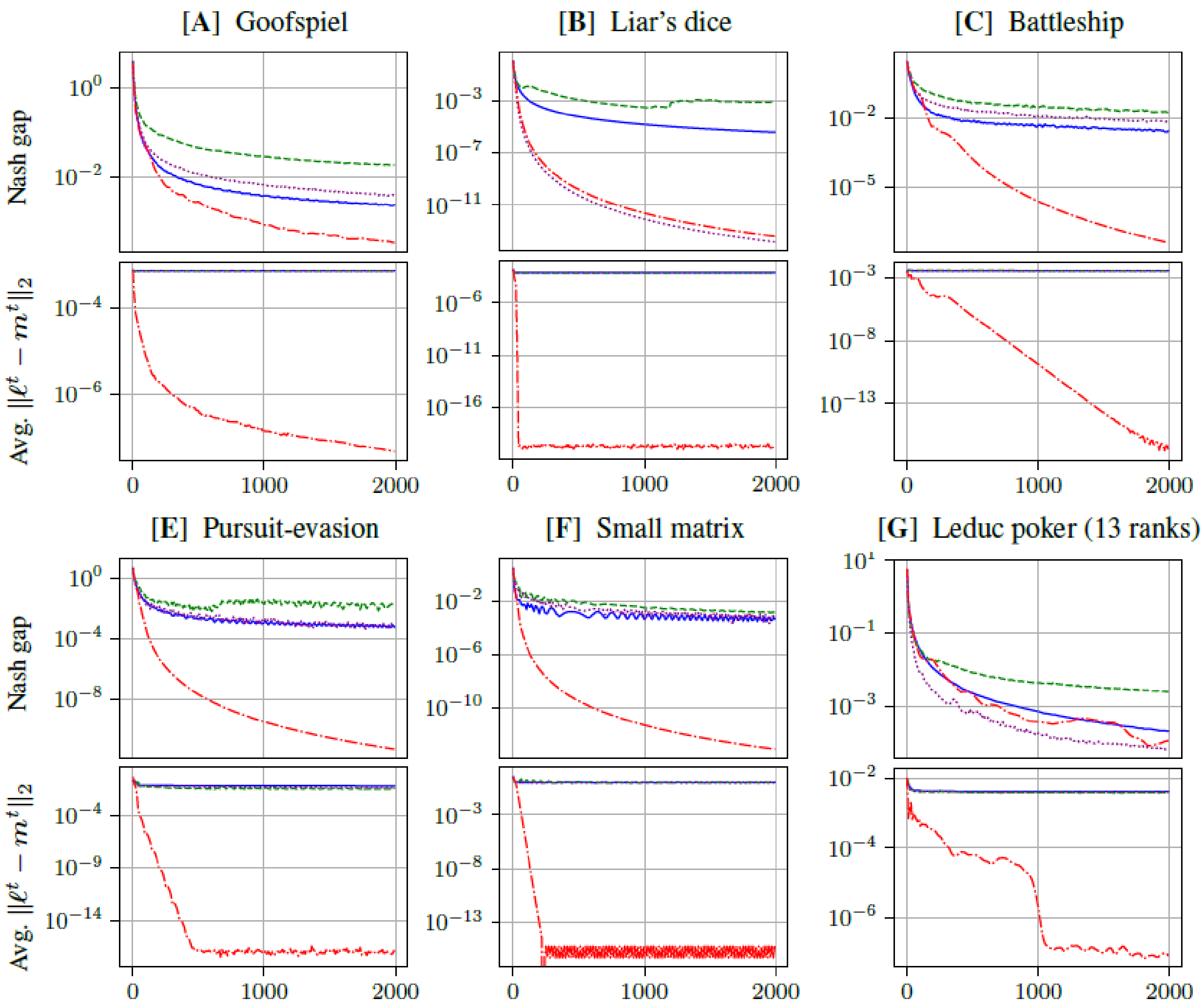
- CFR uses an approach local to each decision point (easier to parallelize, warm-start, etc.)
  - [Brown & Sandholm, Reduced Space and Faster Convergence in Imperfect-Information Games via Pruning. ICML-17]
  - [Brown & Sandholm, Strategy-based warm starting for regret minimization in games, AAAI 2016]
- No need for expensive projections onto feasible strategy polytope (think projected gradient descent)

Other approaches

- Offline first-order methods:
  - e.g., mirror prox (MP) or excessive gap technique (EGT)
    - O(1/T) convergence instead of CFR's O(1 / $\sqrt{T}$)
  - Regret minimization is decentralized, and with optimism it matches the same theoretical rates. Also, it performs better empirically

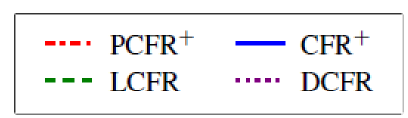
- *All in all, regret-based methods are today the scalable state of the art*

CFR Framework + Predictivity



Dimension of the games

	Decision points	Actions	Leaves
[A]	4.4×10^6	5.3×10^6	1.7×10^6
[B]	2.5×10^4	4.9×10^4	1.5×10^5
[C]	1.1×10^6	3.2×10^6	3.5×10^6
[D]	4.4×10^4	1.3×10^5	5.4×10^7
[E]	1.2×10^4	6.9×10^4	1.2×10^5
[F]	2	4	4
[G]	5.1×10^3	1.2×10^4	9.9×10^4



Important Takeaways

- You can construct a regret minimizer for **sequential** decision making problems by combining regret minimizers for individual decision points

  ⇒ Improvements on simplex domains carry over to extensive-form domains!

- Predictivity works well in extensive-form domains

Techniques to Further Increase Scalability of CFR

- Using utility estimators
  - Similar idea as stochastic gradient descent vs gradient descent
  - Instead of exactly computing the green numbers (gradients of the utility function), we use cheap unbiased estimators
  - Popular estimator: sample a trajectory in the game tree and use importance sampling
  - **“Monte Carlo CFR”**

[Monte Carlo Sampling for Regret Minimization in Extensive Games; Lanctot, Waugh, Zinkevich, Bowling NIPS 2009]

Techniques to Further Increase Scalability of CFR

- Temporary pruning
  - Idea: During CFR traversals,
    - Prune actions (and their subtrees) that have negative cumulative regret
    - Project in how many iterations at the earliest the cumulative regret can turn positive, and catch up that subtree's iterates then
      - » Cath up as if we had played a best response throughout the CFR iterations to the opponent's average strategy in that subtree
      - » If the regret is still negative, re-project, etc.
  - Can be implemented without storing the pruned subtrees => space advantage
  - **Theorem.** In a zero-sum game, if both players choose strategies according to CFR with this kind of pruning, conducting T iterations traverses only

$$O(|S|T + |H| \ln(T)) \text{ nodes}$$

All game paths (i.e., information sets)

Game paths that are part of some best response to some equilibrium

[Brown & Sandholm, Reduced Space and Faster Convergence in Imperfect-Information Games via Pruning. ICML-17]

Techniques to Further Increase Scalability of CFR

- Sparsification
  - Idea: compute a low-rank factorization of the payoff matrix of the game
  - For some games, a low-rank factorization is guaranteed to exist, and can dramatically speed up the algorithms

[Zhang & Sandholm, Sparsified linear programming for zero-sum equilibrium finding. ICML 2020]
[Farina & Sandholm, Fast Payoff Matrix Sparsification Techniques for Structured Extensive-Form Games. AAAI 2022]