6.S890: Topics in Multiagent Learning

Lecture 10 – Prof. Farina

Learning in Extensive-Form Games

Fall 2024

Recall: No-External-Regret

X = Simplex for normal-form games

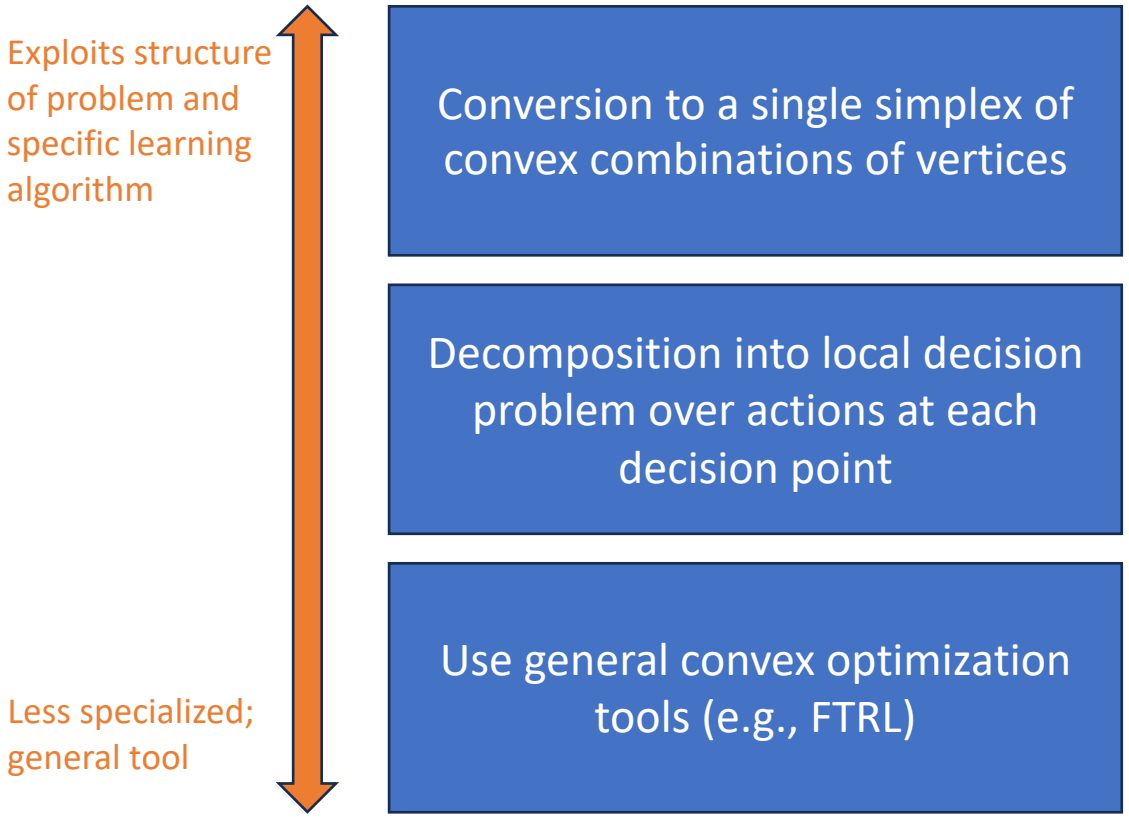
X = sequence-form polytope for extensive-form games

Objective: sublinear (external) regret

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

No-Regret Algorithms for EFGs

Different conceptual approaches exist:



No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

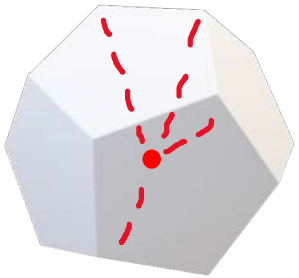
Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Main idea:



Key question:

How to sidestep exponential size?

Every point in the polytope is a convex combination of its finitely many vertices $V := \{v_1, \dots, v_m\}$. So, operate a change of **variable:** learn the convex combination, not the points $x^{(t)}$

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

Perf. of vertex

$$R^{(T)} := \max_{\hat{\lambda} \in \Delta(V)} \sum_{t=1}^{T} \left\langle \begin{pmatrix} \vdots \\ \langle u^{(t)}, v \rangle \\ \vdots \end{pmatrix}, \hat{\lambda} - \lambda^{(t)} \right\rangle$$

No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

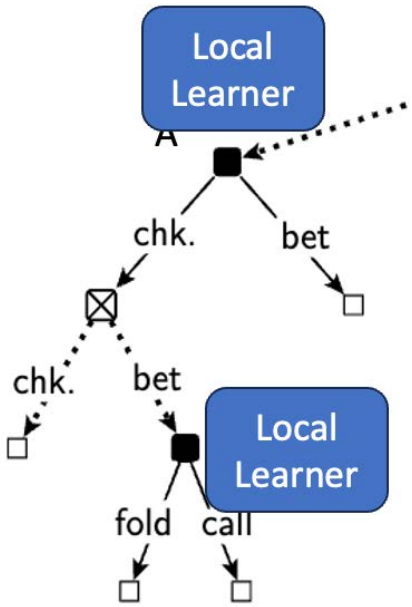
Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Main idea:



Key question:

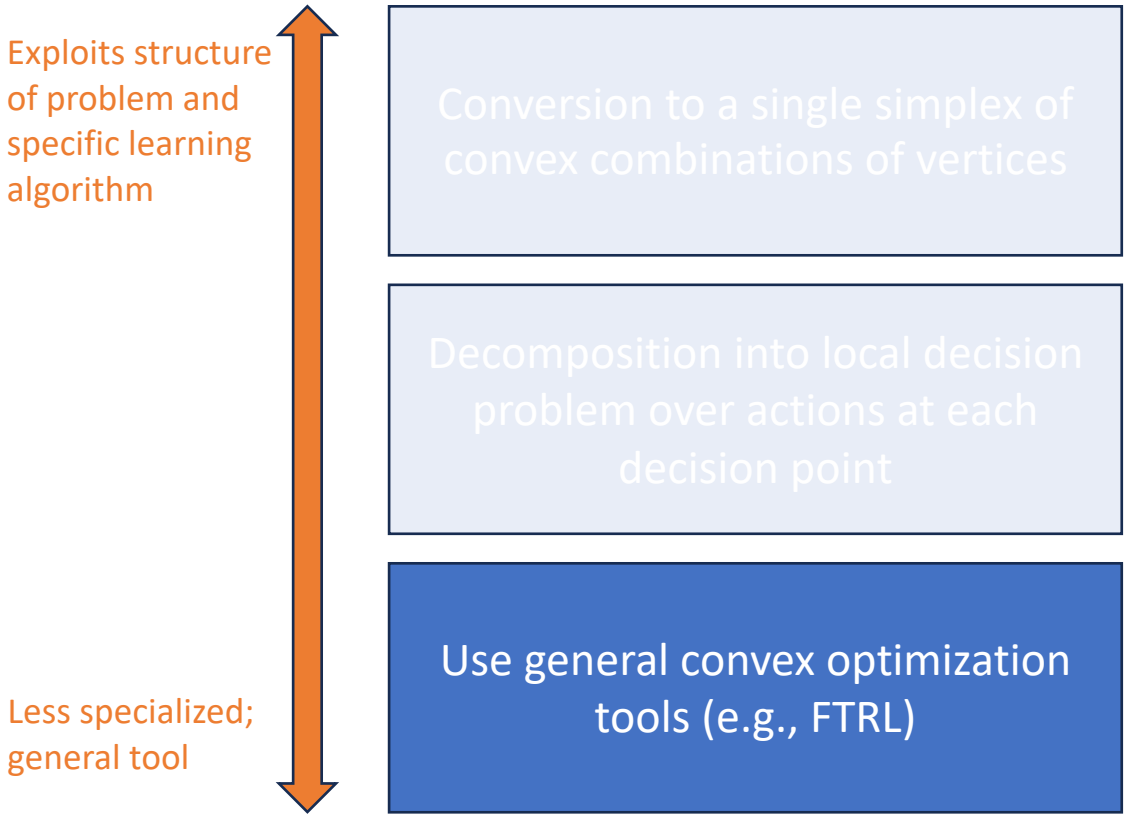
What is the local feedback?

Run a local no-regret algorithm at each decision point to update your strategy.

"Process" the utility vector $u^{(t)}$ (which is for the whole sequence-form strategy) and chop it up into local feedback for each decision point.

No-Regret Algorithms for EFGs

Different conceptual approaches exist:



Main idea:

Key question:

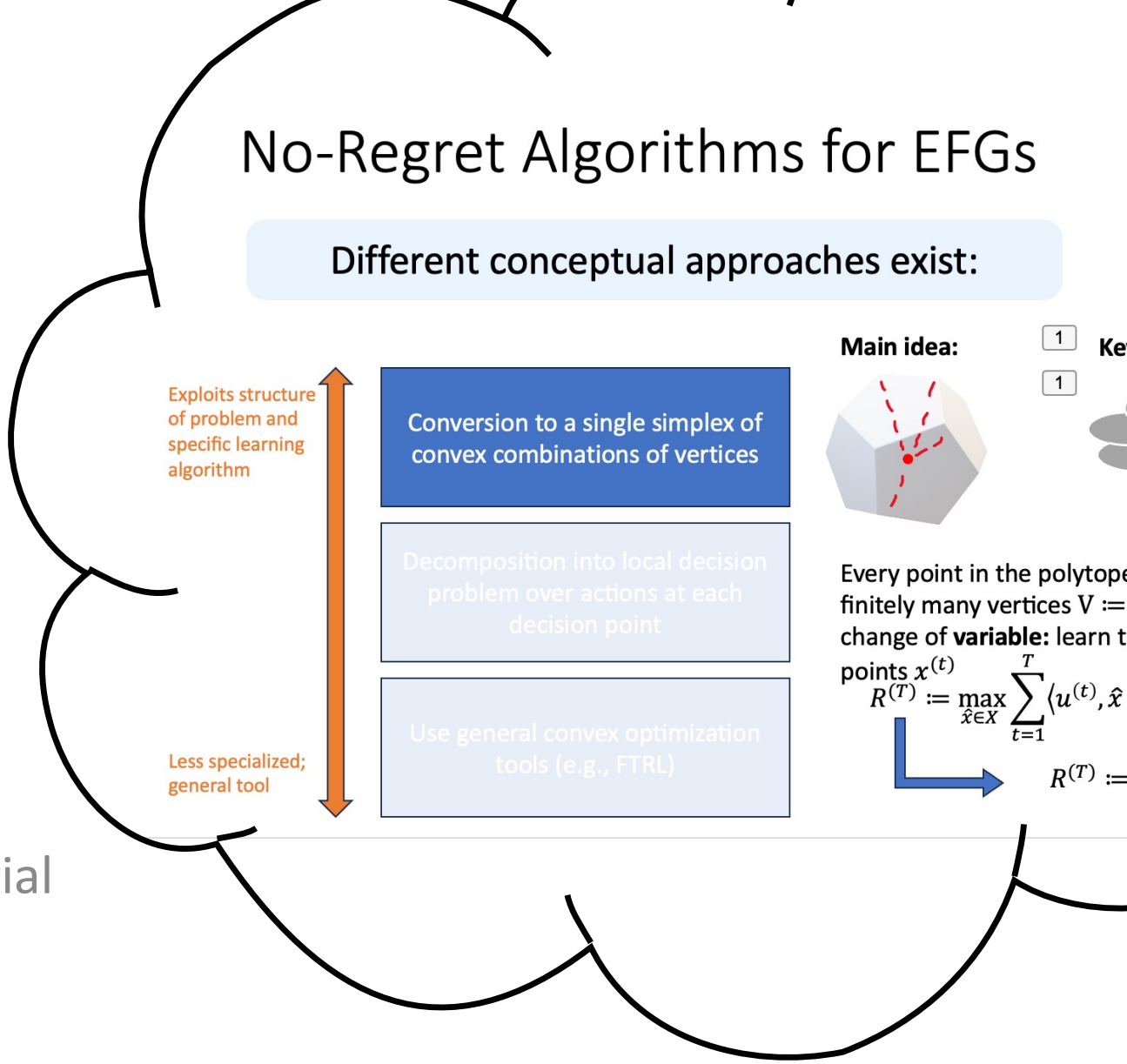
What regularizers are easy to deal with?

The sequence-form polytope is a convex set. So, we can apply the FTRL algorithm in its general form, and that guarantees no-regret

$$x^{(t)} = \arg\max_{x \in Q} \langle U^{(t)}, x \rangle - \frac{1}{\eta}\varphi(x)$$

Kernelized MWU

Applies generically to a set of combinatorial games. We will see this in a later class.



Counterfactual Regret Minimization

No-Regret Algorithms for EFGs

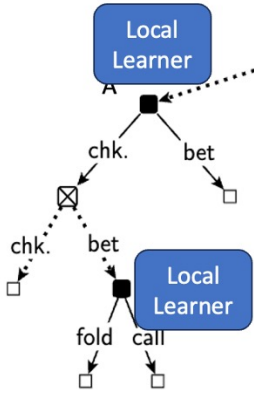
Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

Less specialized; general tool

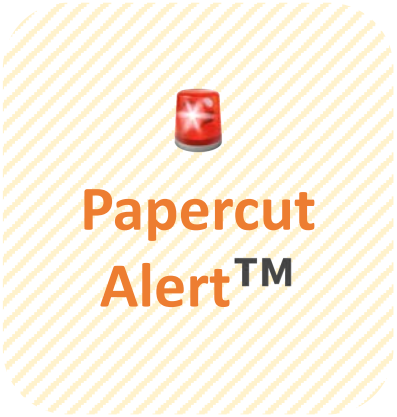
- Conversion to a single simplex of convex combinations of vertices
- **Decomposition into local decision problem over actions at each decision point**
- Use general convex optimization tools (e.g., FTRL)

Main idea:



Counterfactual Regret Minimization

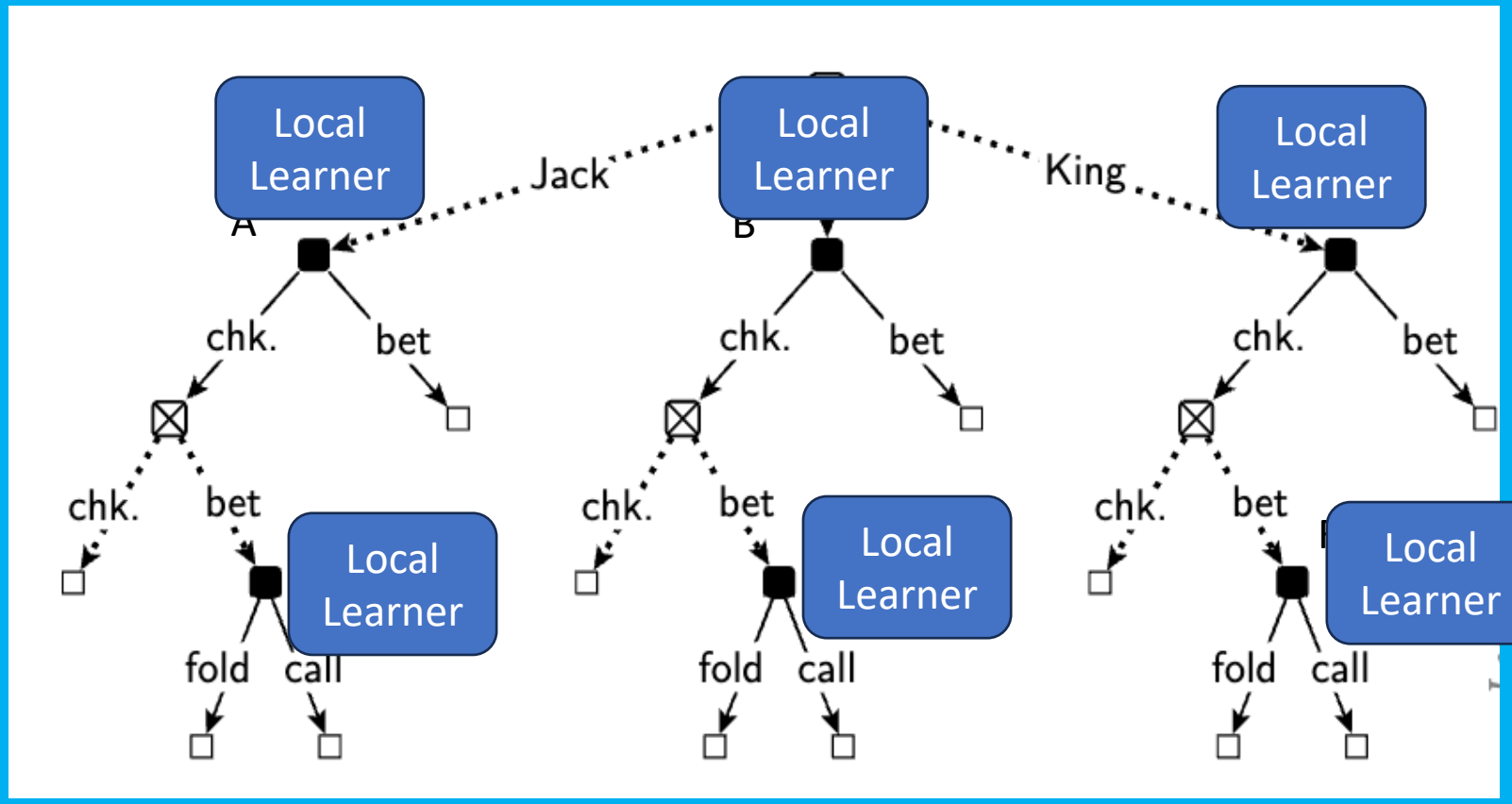
Idea: Minimize regret **globally** on the tree
by **thinking locally** at each decision point



CFR updates strategies in *behavioral* form…

…but is a no-external-regret algorithm for
sequence-form strategies

Big Picture Idea:



Each local learner is responsible for refining the **behavior** at their decision point

Can locally use regret matching, multiplicative weights update, …

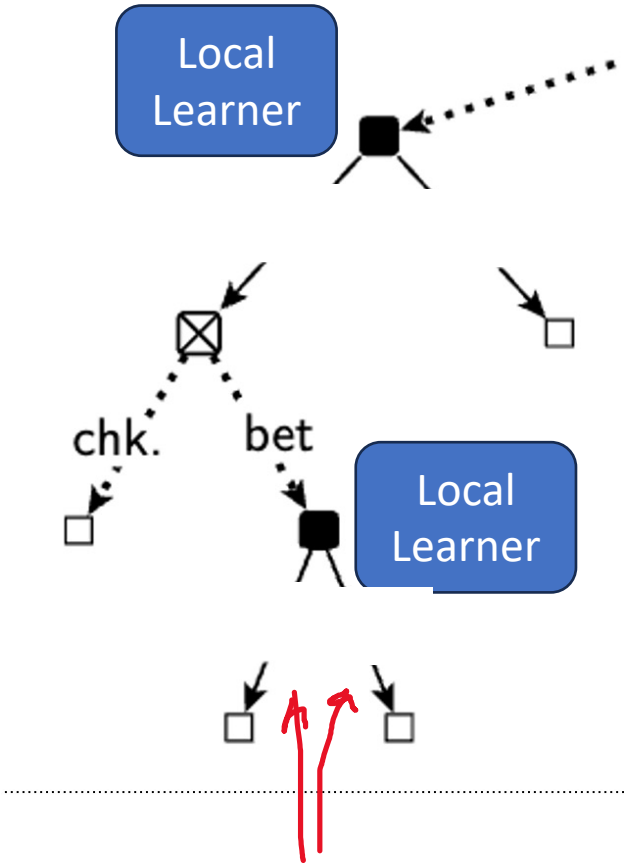
Local Training Feedback

Each local learner receives as feedback what is known as a **counterfactual utility vector**

This is constructed starting from the $u^{(t)}$

Recall: Learning in Normal-Form Games

Recall: Learning in Normal-Form Games



Probabilities of actions chosen by local learners

CFR Learning Algorithm

b_1

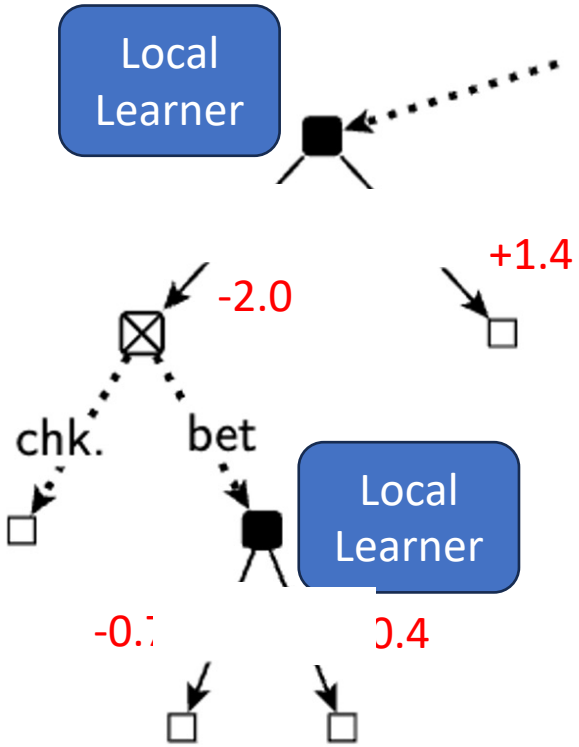
b_2

$b_1 b_3$

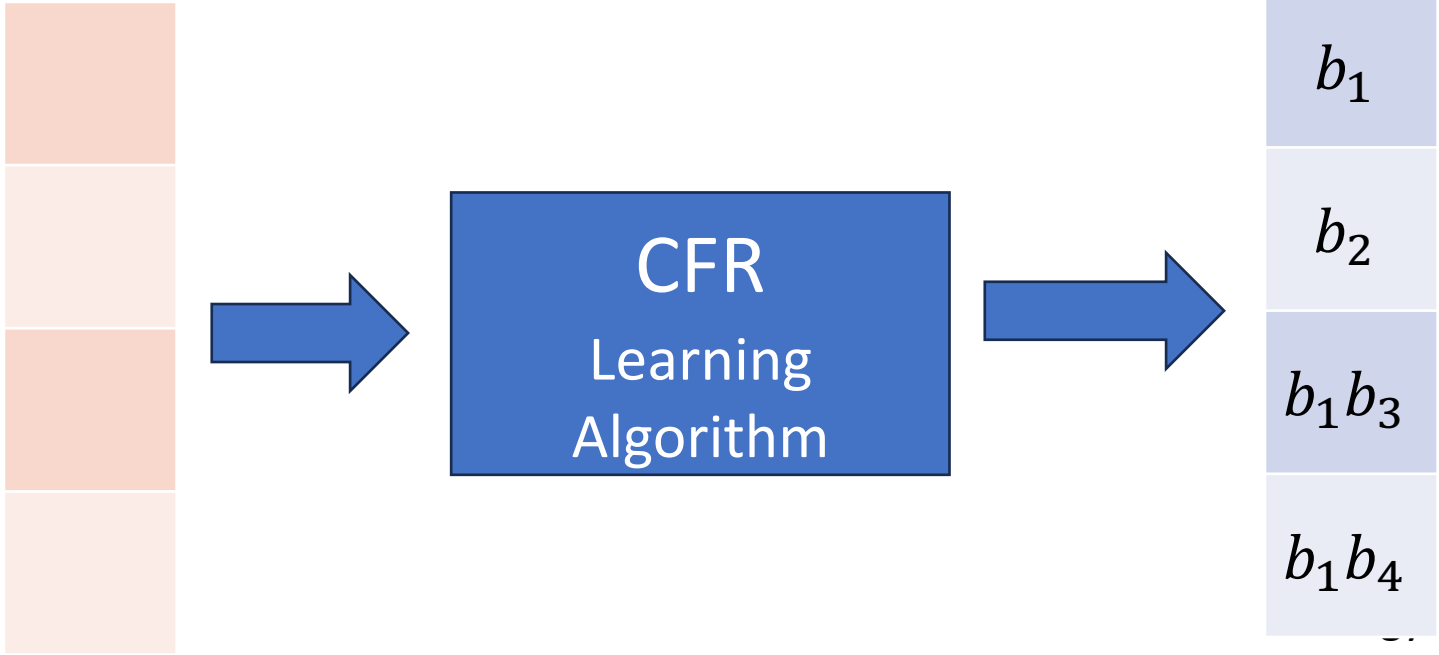
$b_1 b_4$

Strategy (in sequence form)

Recall: Learning in Normal-Form Games



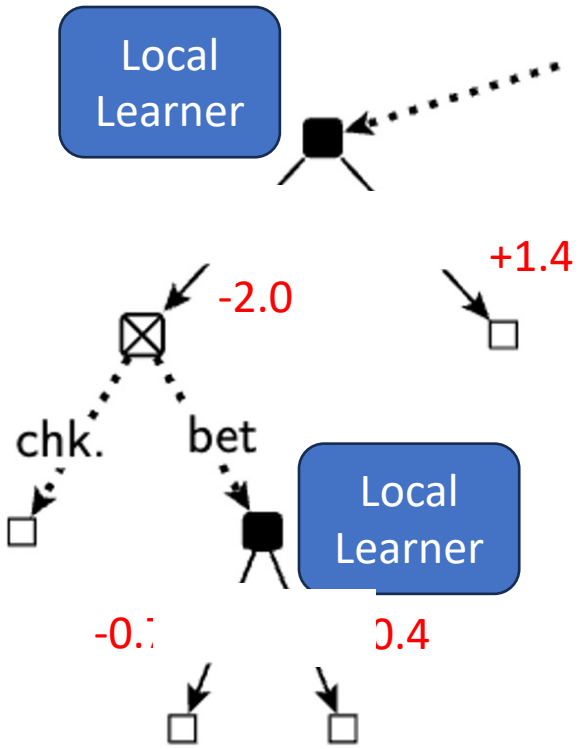
Main question: what utility to pass to the local learners?



Utility vector (for sequence-form strategy)

Strategy (in sequence form)

Counterfactual Utilities



Give to each local learner the **expected utility in the subtree** rooted at each action:

$$\widehat{u_3} = -0.7$$
$$\widehat{u_4} = -0.4$$
$$\widehat{u_2} = +1.4$$
$$\widehat{u_1} = -2.0 + b_3 \cdot (-0.7) + b_4 \cdot (-0.4)$$

Regret bound

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

$$R_{CFR}^{(T)} \leq \sum_{j} \max\left\{0, R_j^{(T)}\right\}$$

Decision points

Local regret cumulated by learner at j

- **Therefore**: if the local regret minimizers all have regret $O(\sqrt{T})$, then CFR has regret $O(\sqrt{T})$ (where the O hides game-dependent constants)

Therefore: if both players in a zero-sum extensive-form game play according to CFR, the average strategy converges to Nash equilibrium at rate $O(1/\sqrt{T})$

Implementation details

- Producing strategies is easy: query all local learners and construct the sequence-form strategy

Algorithm 1: CFR regret minimizer

Data: $\mathcal{R}_j$ regret minimizer for $\Delta(A_j)$; one for each decision point $j \in \mathcal{J}$ of the TFDP.

function NEXTSTRATEGY()
  [▷ Step 1: we ask each of the $\mathcal{R}_j$ for their next strategy local at each decision point]
  for each decision point $j \in \mathcal{J}$
    $b_j^{(t)} \in \Delta(A_j) \leftarrow \mathcal{R}_j$.NEXTSTRATEGY()
  [▷ Step 2: we construct the sequence-form representation of the strategy that plays according to the distribution $b_j^{(t)}$ at each decision point $j \in \mathcal{J}$]
  $x^{(t)} = \mathbf{0} \in \mathbb{R}^{\Sigma}$
  for each decision point $j \in \mathcal{J}$ in *top-down traversal* order in the TFDP
    for each action $a \in A_j$
      if $p_j = \varnothing$
        $x^{(t)}[ja] \leftarrow b_j^{(t)}[a]$
      else
        $x^{(t)}[ja] \leftarrow x^{(t)}[p_j] \cdot b_j^{(t)}[a]$
  [▷ You should convince yourself that the vector $x^{(t)}$ we just filled in above is a valid sequence-form strategy, that is, it satisfies the required consistency constraints we saw in Lecture 9. In symbols, $x^{(t)} \in Q$]
  return $x^{(t)}$

Implementation details

- When feedback arrives, construct counterfactual utilities. It can be done with a **single bottom up traversal**!

12 **function** ObserveUtility($u^{(t)} \in \mathbb{R}^{\Sigma}$)

[▷ Step 1: we compute the expected utility for each subtree rooted at each node $v \in \mathcal{J} \cup \mathcal{K}$]

13 $V^{(t)} \leftarrow$ empty dictionary [▷ eventually, it will map keys $\mathcal{J} \cup \mathcal{K} \cup \{\bot\}$ to real numbers]

14 $V^{(t)}[\bot] \leftarrow 0$

15 **for each** node in the tree $v \in \mathcal{J} \cup \mathcal{K}$ in *bottom-up traversal* order in the TFDP

16 **if** $v \in \mathcal{J}$

17 let $j \leftarrow v$

18 $V^{(t)}[j] \leftarrow \sum_{a \in A_j} b_j^{(t)}[a] \cdot \left(u^{(t)}[ja] + V^{(t)}[\rho(j,a)]\right)$

19 **else**

20 let $k \leftarrow v$

21 $V^{(t)}[k] \leftarrow \sum_{s \in S_k} V^{(t)}[\rho(k,s)]$

[▷ Step 2: at each decision point $j \in \mathcal{J}$, we now construct a local utility vector $u_j^{(t)}$ called *counterfactual utility*]

22 **for each** decision point $j \in \mathcal{J}$

23 $u_j^{(t)} \leftarrow \mathbf{0} \in \mathbb{R}^{A_j}$

24 **for each** action $a \in A_j$

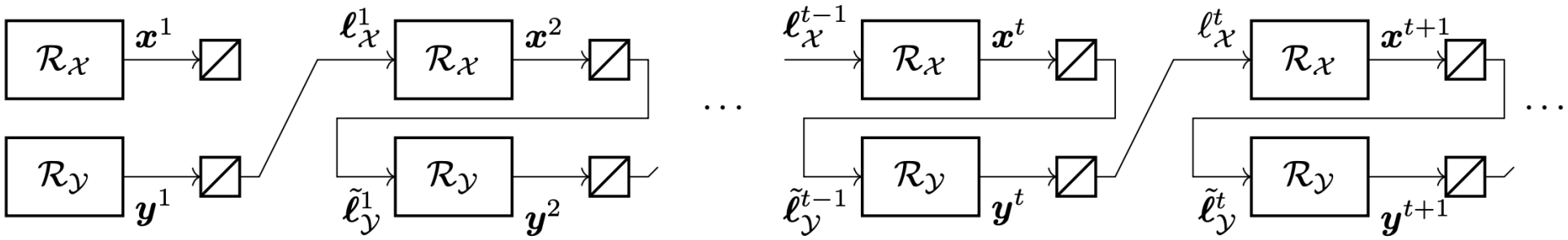
25 $u_j^{(t)}[a] \leftarrow u^{(t)}[ja] + V^{(t)}[\rho(j,a)]$

26 $\mathcal{R}_j$.ObserveUtility($u_j^{(t)}$)

Further pushing performance

CFR+: CFR with the following settings:

- Regret Matching+ at each decision point (see Lecture 5)
- Use alternation



- When computing average strategy, weigh strategy at time t by t:

$$\bar{x}^{(T)} \propto \sum_{19}^{T} t \cdot x^{(t)}$$

Advantages of CFR

Compared to linear programming, CFR is significantly more scalable

…On the other hand, it converges to equilibrium at a 1/sqrt(T) rate, rather than e^(-T)

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]
- …

CFR Lends itself to further extensions

- 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]
  - Even better algorithm, **ESCHER**, does not use importance sampling [McAleer, Farina, Lanctot & Sandholm *ICLR*-23]

6.S890 Topics in Multiagent Learning

Fall 2024