# 6.S890:
# Topics in Multiagent Learning

Lecture 14 – Prof. Farina

**Learning in Extensive-Form Games**

Fall 2023

# Game Solving

**Approach 1**: Linear Programming
(two-player zero-sum)

Last time

**Approach 2**: Learning

Today

# Recall: No-External-Regret

$X$ = Simplex for normal-form games

$X$ = sequence-form polytope for extensive-form games

$u^{(t)}$ Utility vectors → **Learning Algorithm** → Strategies $x^{(t)} \in X$

Objective: sublinear (external) regret

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

# Recall: Learning in Normal-Form Games

# Recall: Learning Algorithms

Regret matching (RM): Probability of each action proportional to ReLU of regret on the action

$$x^{(t)} \propto \left[r^{(t)}\right]^+$$

Multiplicative Weights Update (MWU): Prob. of each action proportional to exp of regret on the action

$$x^{(t)} \propto \exp(\eta \cdot r^{(t)})$$

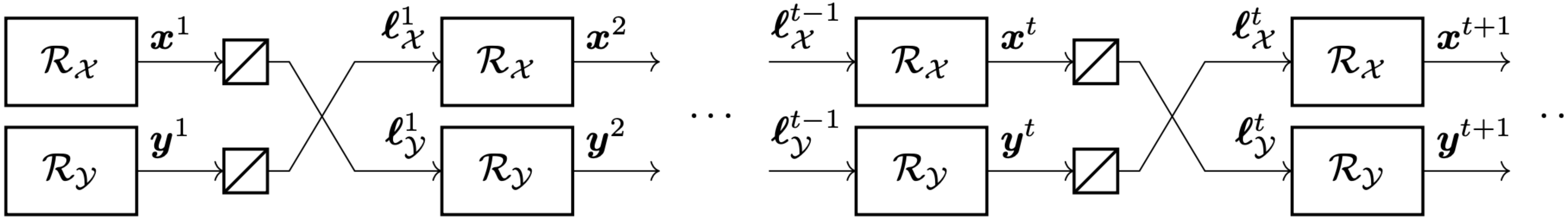
Recall (HW1): MWU is FTRL with negative entropy

Follow-The-Regularized-Leader (FTRL):

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

# Recall: Connections with Equilibria

- Recall: when all players play external-regret-minimizing strategies, then:
  - In two-player zero-sum games, their average strategies converge to the set of Nash equilibrium (gives an alternative approach to previous lecture)
  - In general, the average product distribution of play converges to the set of coarse-correlated equilibria




$$\boldsymbol{\ell}_\mathcal{X}^t := \boldsymbol{A}\boldsymbol{y}^t, \qquad \boldsymbol{\ell}_\mathcal{Y}^t := -\boldsymbol{A}^\top \boldsymbol{x}^t$$

# 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

# No-Regret Algorithms for EFGs

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

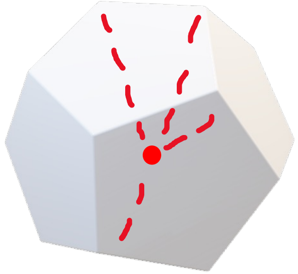
Conversion to a single simplex of convex combinations of vertices

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 $V := \{v_1, \ldots, v_m\}$. So, operate a change of **variable:** learn the convex combination, not the points $x^{(t)}$

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

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

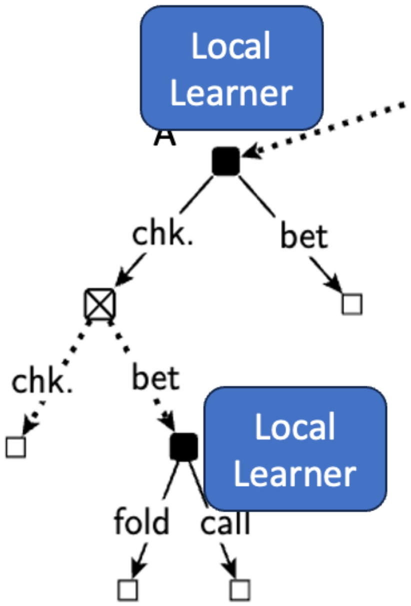
Conversion to a single simplex of convex combinations of vertices

**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:

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:**

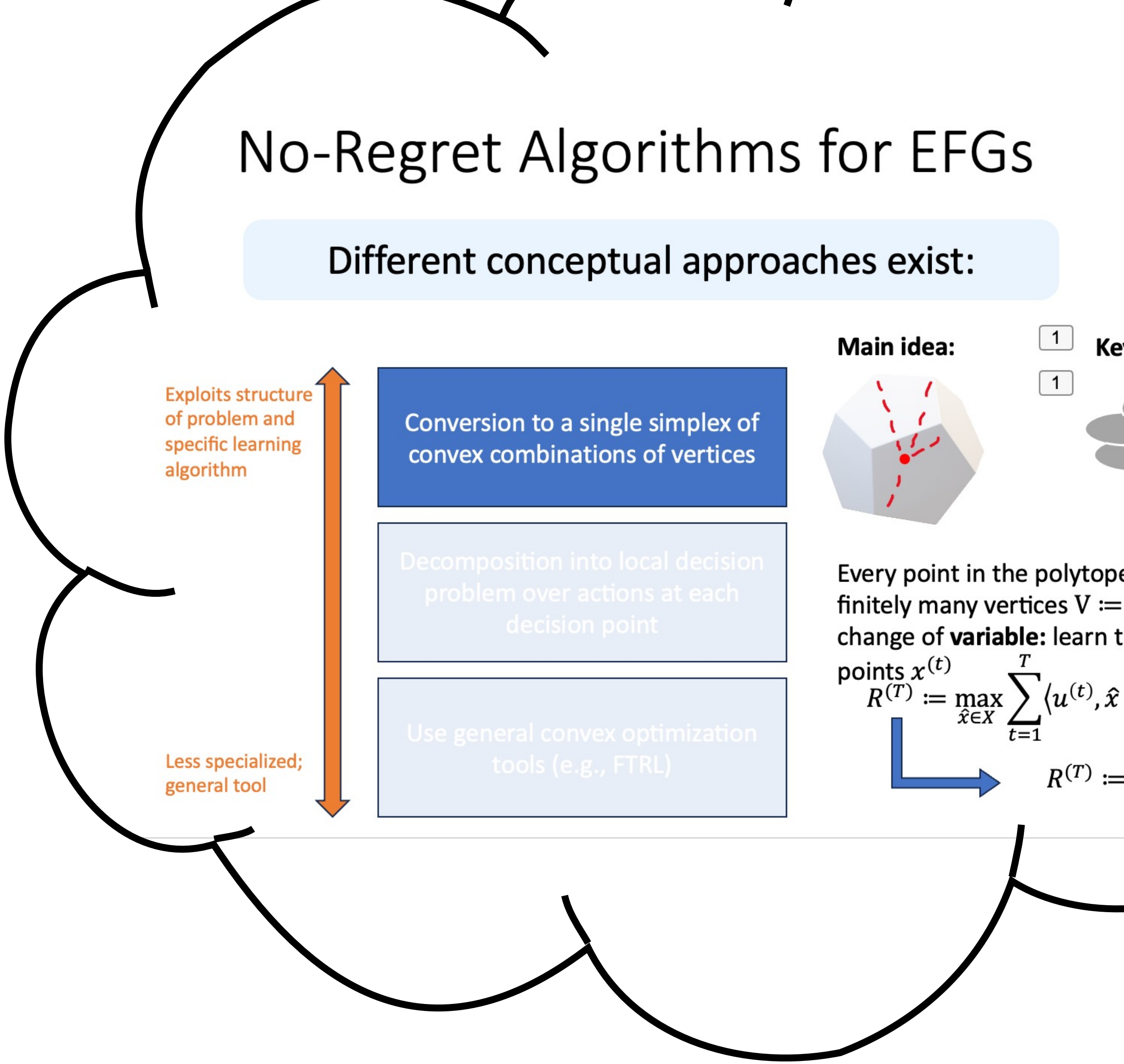
The sequence-form polytope is a convex set. So, we can apply the FTRL algorithm in its general form, and that guarantees no-regret

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

**Key question:**

What regularizers are easy to deal with?

# Kernelized MWU



***General Setup:***

$\Omega_{\mathrm{i}} \subseteq \mathbb{R}^d$ polyhedral strategy set for Player i (e.g., sequence-form polytope for EFGs) with 0/1 vertices

$V_i$ vertices of $\Omega_{\mathrm{i}}$

## **Vertex MWU** algorithm

***Setup***
$\Omega_{\mathrm{i}} \subseteq \mathbb{R}^d$
$V_i$ vertices of $\Omega_{\mathrm{i}}$

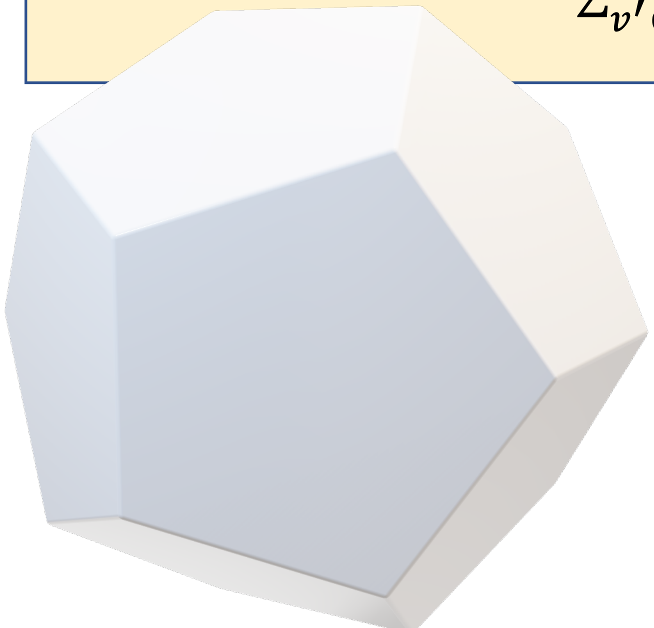
$\lambda^{(1)} \coloneqq \frac{1}{|V_i|}\mathbf{1} \in \mathbb{R}^{V_i}$

For $t = 1, 2, \ldots$

Play mixed strategy $\Omega_i \ni x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe reward vector $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v \rangle}}{\sum_{v' \in V_i} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v' \rangle}}$



“Utility of vertex v”

…We weight vertices using MWU

### *Main theorem*

When $\Omega_i$ has 0/1-coordinate vertices, Vertex MWU can be implemented using *d+1* evaluations of the 0/1-polyhedral kernel at each iteration

## **Vertex MWU** algorithm

***Setup***

$\Omega_\mathrm{i} \subseteq \mathbb{R}^d$

$V_i$ vertices of $\Omega_\mathrm{i}$

$\lambda^{(1)} := \frac{1}{|V_i|}\mathbf{1} \in \mathbb{R}^{V_i}$

For $t = 1, 2, \ldots$

Play mixed strategy $\Omega_i \ni x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe reward vector $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] := \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V_i} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

Crucially independent on the number of vertices of $\Omega_i$!

As long as the kernel function can be evaluated efficiently, then Vertex (O)MWU can be simulated in polynomial time

**Setup**

$\Omega \subseteq \mathbb{R}^d$

$V$ vertices of $\Omega$

$V \subseteq \{0,1\}^d$

***Definition*** (0/1-feature map of $\Omega$)

$$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \qquad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$$

Given any vector, for each vertex it computes the product of the coordinates that are hot for that vertex

***Definition*** (0/1-polyhedral kernel of $\Omega$)

$$K_\Omega : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \quad K_\Omega(x,y) \coloneqq \langle \phi_\Omega(x), \phi_\Omega(y) \rangle = \sum_{v \in V} \prod_{k:v[k]=1} x[k] \cdot y[k]$$

# Let’s see how the feature map and the kernel help simulate Vertex MWU

$$\boldsymbol{\lambda}^{(t)} = \frac{\phi_\Omega(\boldsymbol{b}^{(t)})}{K_\Omega(\boldsymbol{b}^{(t)}, \mathbf{1})}$$

# Idea #1

Recall (feature map):

$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \quad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$

***Lemma 1:*** At all times t, $\lambda^{(t)}$ is proportional to the feature map of the vector

$$\mathbb{R}^d \ni b^{(t)} \coloneqq \exp\left\{\eta \sum_{\tau=1}^{t-1} u^{(\tau)}\right\}$$

Proof: by induction

## Vertex MWU algorithm

***Setup***

$\Omega \subseteq \mathbb{R}^d$

$V$ vertices of $\Omega$

$V \subseteq \{0,1\}^d$

$\lambda^{(1)} \coloneqq \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

3 Play $x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta\,\langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta\,\langle u^{(t)}, v'\rangle}}$

***Consequence:*** by keeping track of $b^{(t)}$ we are implicitly keeping track of $\lambda^{(t)}$ as well

…So, no need to actually perform the update on line 5 explicitly

# Idea #1

Recall (feature map):
$\phi_\Omega : \mathbb{R}^d \to \mathbb{R}^V, \quad \phi_\Omega(x)[v] \coloneqq \prod_{k:v[k]=1} x[k]$

***Lemma 1:*** At all times t, $\lambda^{(t)}$ is proportional to the feature map of the vector

$\left( \quad t-1 \quad \right)$

Pr

Remaining obstacle: how can we evaluate line 3 with only implicit access to $\lambda^{(t)}$ via $b^{(t)}$?

## **Vertex MWU** algorithm

***Setup***
$\Omega \subseteq \mathbb{R}^d$
$V$ vertices of $\Omega$
$V \subseteq \{0,1\}^d$

$\lambda^{(1)} \coloneqq \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

3 Play $x^{(t)} \coloneqq \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] \coloneqq \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

***Consequence:*** by keeping track of $b^{(t)}$ we are implicitly keeping track of $\lambda^{(t)}$ as well

…So, no need to actually perform the update on line 5 explicitly

# Idea #2

***Lemma 1:*** At all times t, $\lambda^{(t)}$ is proportional to the feature map of the vector

$$\mathbb{R}^d \ni b^{(t)} := \exp\left\{\eta \sum_{\tau=1}^{t-1} u^{(\tau)}\right\}$$

## **Vertex MWU** algorithm

***Setup***

$\Omega \subseteq \mathbb{R}^d$

$V$ vertices of $\Omega$

$V \subseteq \{0,1\}^d$

$\lambda^{(1)} := \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

3 Play $x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

5 Set $\lambda^{(t+1)}[v] := \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

***Lemma 2:*** At all times t, $x^{(t)}$ can be reconstructed from $b^{(t)}$ as

$$x^{(t)} = \left(1 - \frac{K_\Omega(b^{(t)}, \mathbf{1} - e_1)}{K_\Omega(b^{(t)}, \mathbf{1})}, \ldots, 1 - \frac{K_\Omega(b^{(t)}, \mathbf{1} - e_d)}{K_\Omega(b^{(t)}, \mathbf{1})}\right)$$

($d$+1 kernel evaluations)

## **Vertex MWU** algorithm

***Setup***
$\Omega \subseteq \mathbb{R}^d$
$V$ vertices of $\Omega$
$V \subseteq \{0,1\}^d$

$\lambda^{(1)} := \frac{1}{|V|}\mathbf{1} \in \mathbb{R}^V$

For $t = 1, 2, \ldots$

Play $x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$

Observe utility $u^{(t)} \in \mathbb{R}^d$

Set $\lambda^{(t+1)}[v] := \dfrac{\lambda^{(t)}[v] \cdot e^{\eta \langle u^{(t)}, v\rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle u^{(t)}, v'\rangle}}$

## **Kernelized MWU** algorithm

***Setup***
$\Omega \subseteq \mathbb{R}^d$
$V$ vertices of $\Omega$
$V \subseteq \{0,1\}^d$

$b^{(1)} := \mathbf{1} \in \mathbb{R}^d$

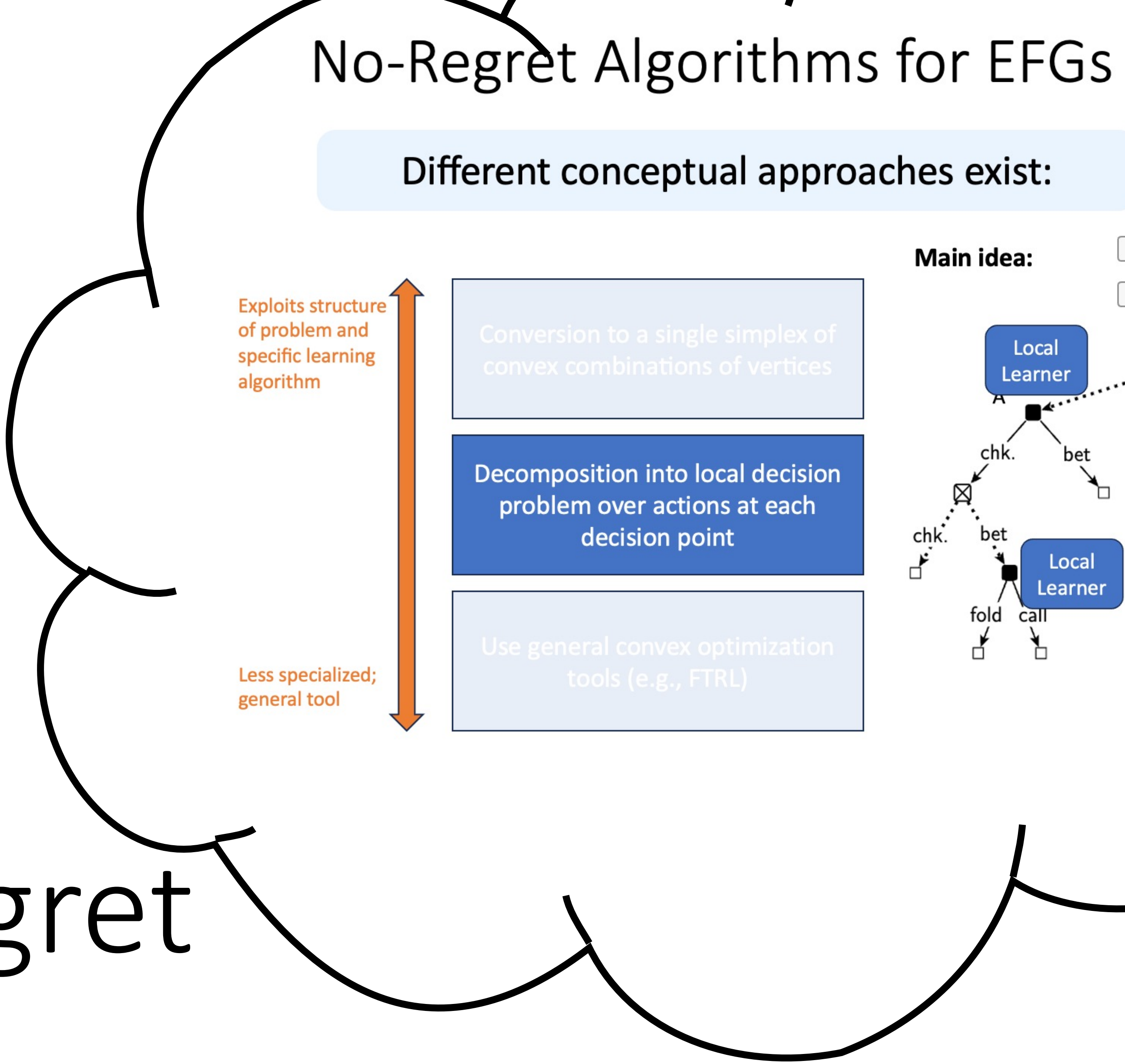
For $t = 1, 2, \ldots$

Play $x^{(t)} := \left(1 - \frac{K_\Omega(b^{(t)}, \mathbf{1} - e_1)}{K_\Omega(b^{(t)}, \mathbf{1})}, \ldots, 1 - \frac{K_\Omega(b^{(t)}, \mathbf{1} - e_d)}{K_\Omega(b^{(t)}, \mathbf{1})}\right)$

Observe utility $u^{(t)} \in \mathbb{R}^d$

Set $b^{(t+1)} := \exp\{\eta \sum_{\tau=1}^{t} u^{(\tau)}\}$

# Counterfactual Regret Minimization



# Counterfactual Regret Minimization

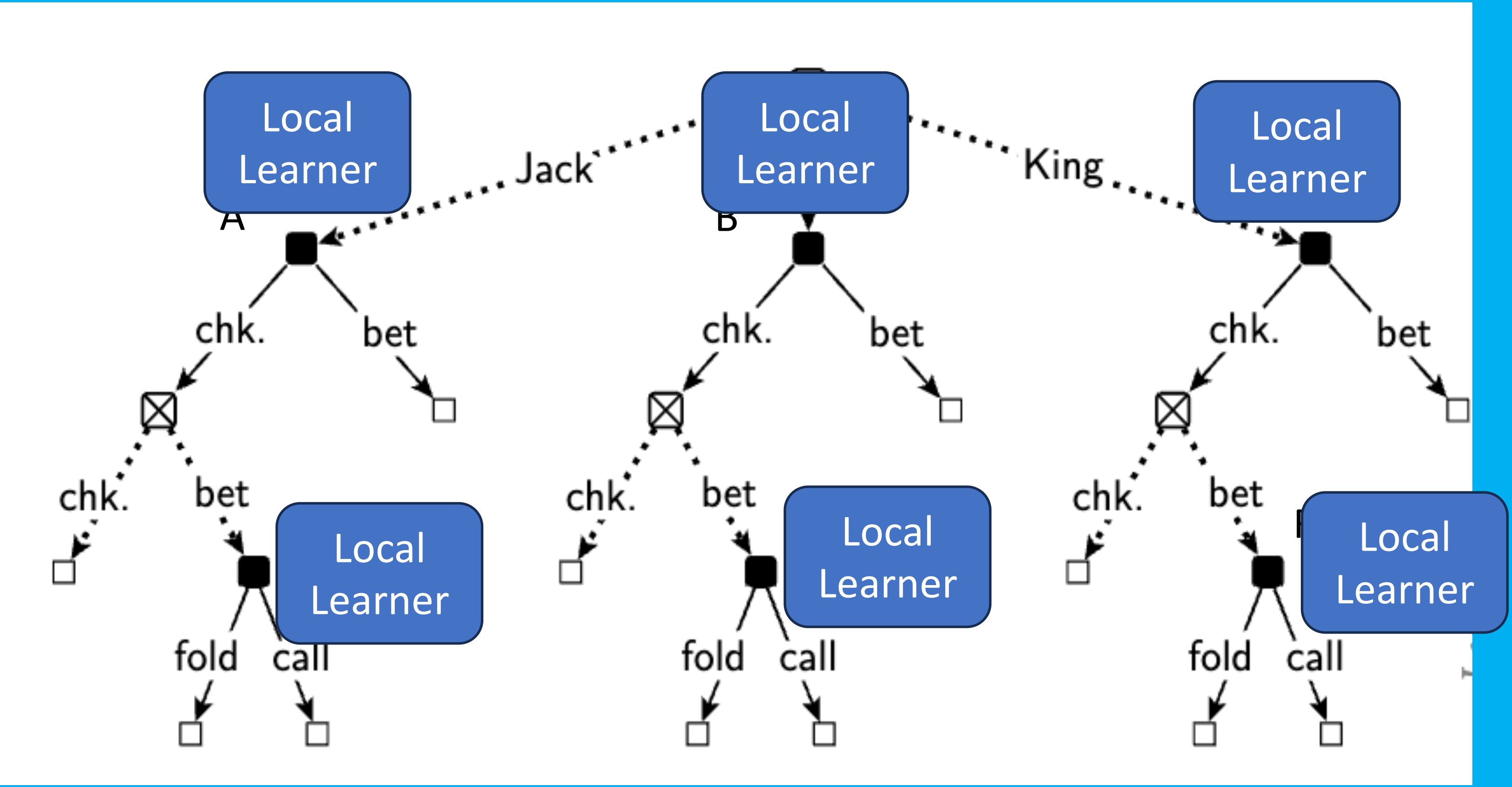
Idea: Minimize regret **globally** on the tree
by **thinking locally** at each decision point

🚨 **Papercut Alert™**

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, …

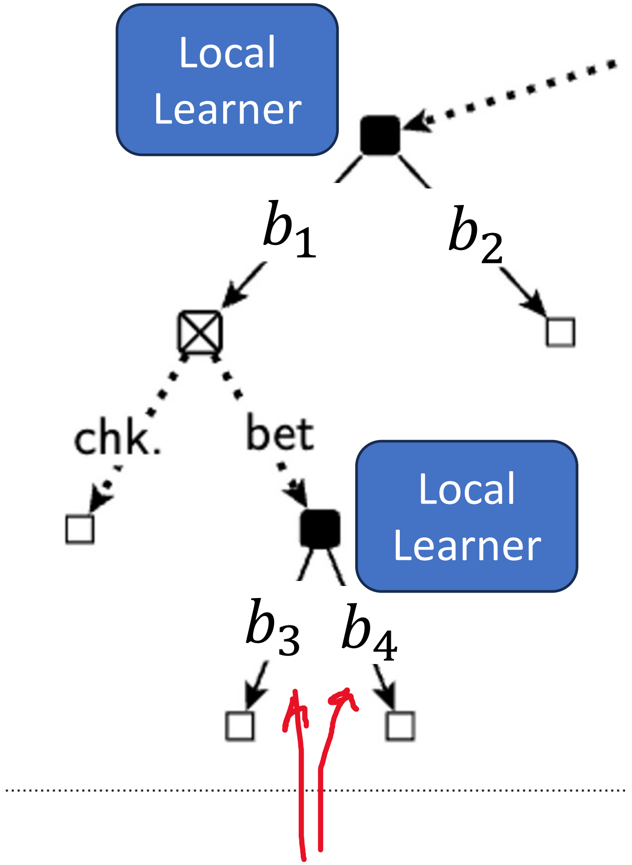
# Local Training Feedback

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

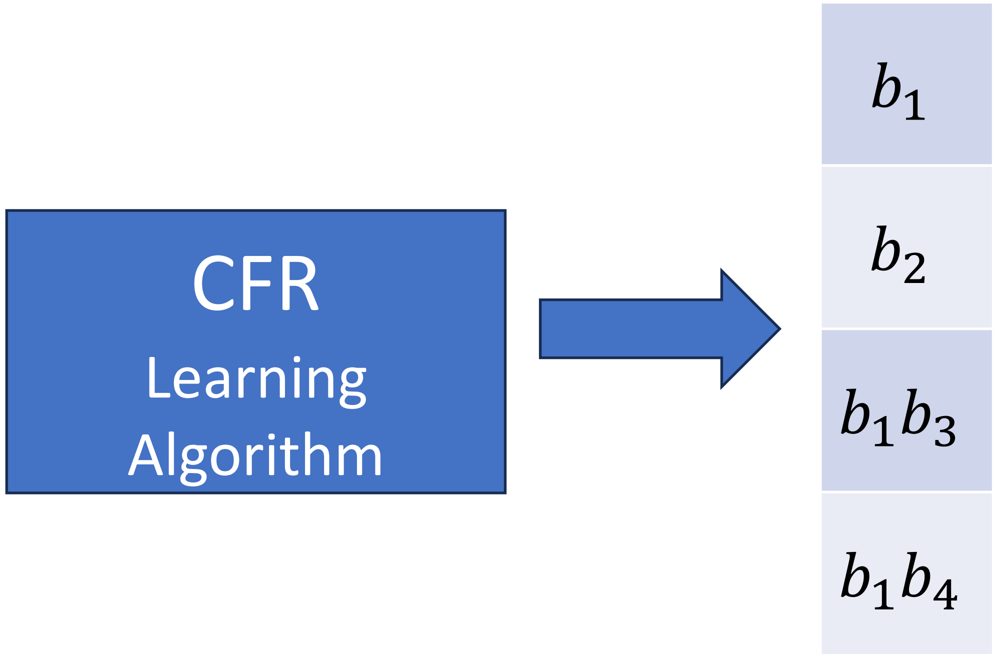
This is constructed starting from the $u^{(t)}$

# Recall: Learning in Normal-Form Games

# Recall: Learning in Normal-Form Games

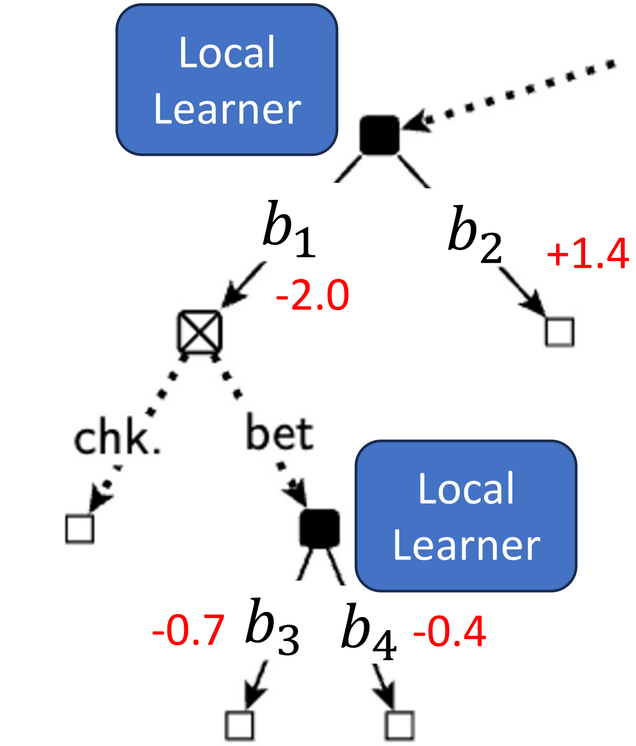



Probabilities of actions chosen by local learners

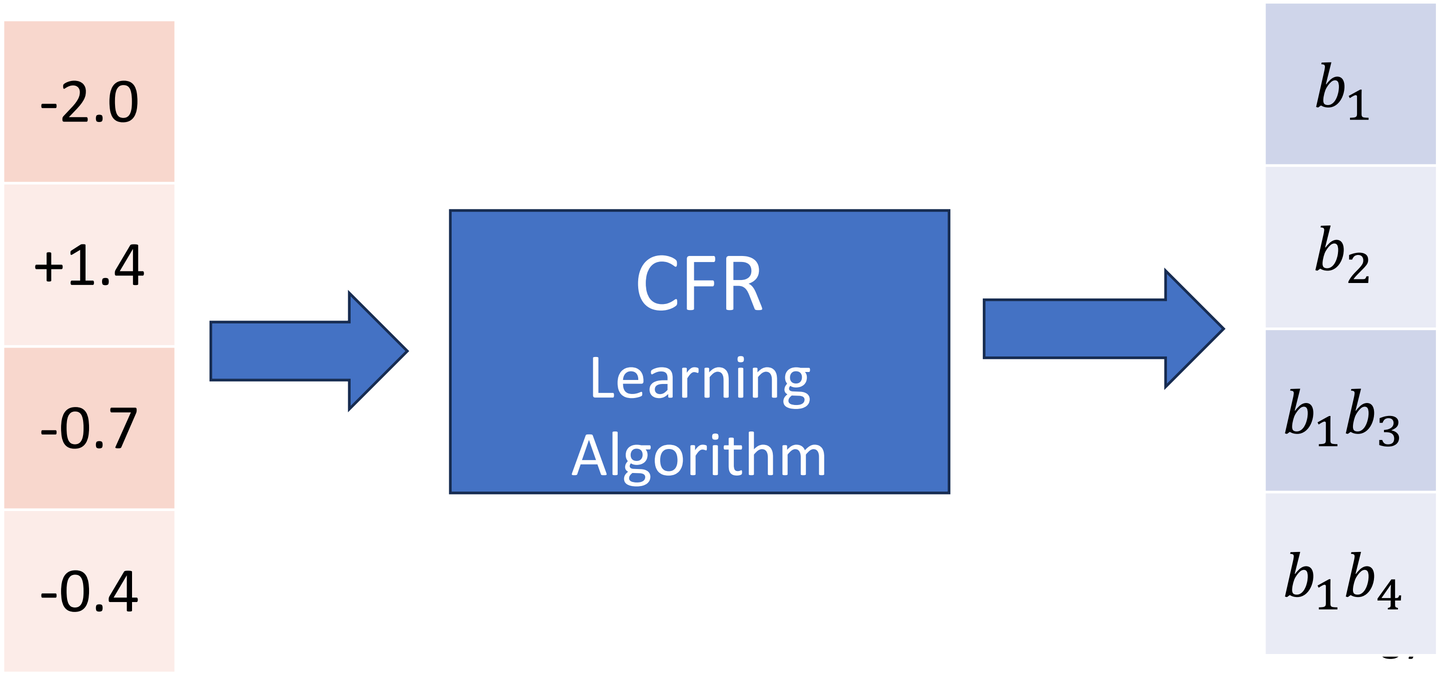



Strategy
(in sequence form)

# Recall: Learning in Normal-Form Games



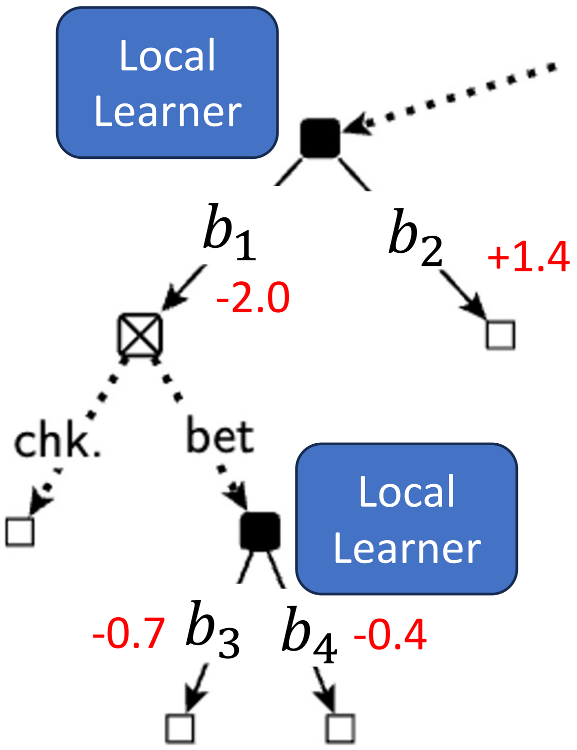

Main question: what utility to pass to the local learners?




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:

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

# Why does it work?

- Proof time!

# Regret bound

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

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

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 $O(1/\sqrt{T})$

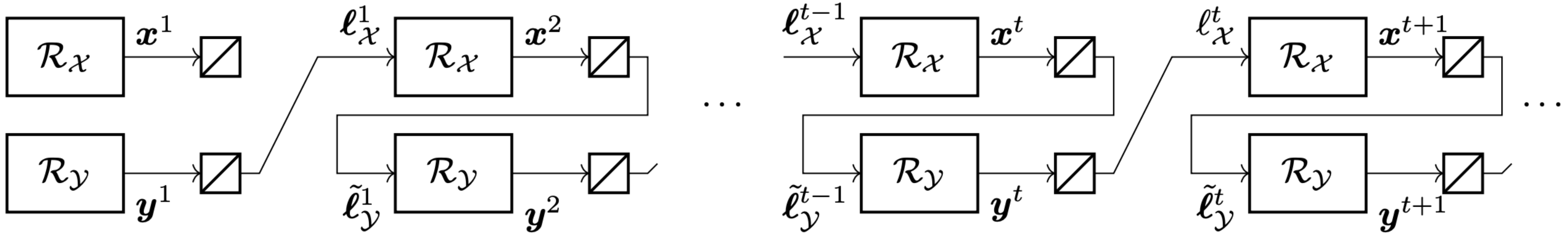
# Implementation details

- See accompanying notes

# 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:

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

# 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]