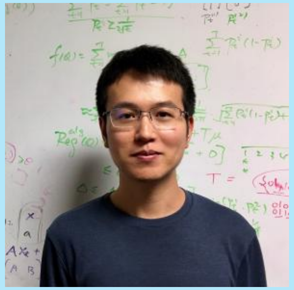
Kernelized Multiplicative Weights for 0/1-Polyhedral Games

Bridging the Gap Between
Learning in Extensive-Form and Normal-Form Games


Gabriele Farina
CMU


Chung-Wei Lee
USC


Haipeng Luo
USC


Christian Kroer
Columbia

No-regret learning in the context of normal-form games (NFGs) has been studied extensively

			
	0	-1	+1
	+1	0	-1
	-1	+1	0

Landmark result in theory of learning in games:

When all players learn using no-regret dynamics (e.g., MWU), the empirical frequency of play converges to the set of coarse correlated equilibria

Even more, in two-player zero-sum games, the average strategies converge to the set of Nash equilibria

As of today, learning is *by far* the most scalable way of computing game-theoretic solutions and equilibria in large games

1. *Linear time strategy updates*
2. *Each agent learns in parallel*
3. *Can often be implemented in a decentralized way*

Over the past decade, faster and faster no-regret dynamics have been developed for normal-form games

★ Most studied algorithm as of today: ***Optimistic Multiplicative Weights Update (OMWU)***

- Per-player regret bound:
  - ☑ Polylog dependence on the number of actions
  - ☑ Polylog(T) dependence on time

Implies $\tilde{O}\left(\frac{1}{T}\right)$ convergence to coarse correlated equilibrium in self-play

[Daskalakis et al. '21]

- Sum of players' regrets
  - ☑ Polylog dependence on #actions
  - ☑ Constant dependence on time

Implies $O\left(\frac{1}{T}\right)$ convergence to Nash eq. in two-player zero-sum games

[Syrgkanis et al. '15]

- ☑ Last-strategy convergence* (2pl 0sum)

[Hsieh et al. '21; Wei et al. '21]

However, normal-form games are a *rather limited* model of strategic interaction

All players act *once* and *simultaneously*

No sequential actions

No observations about other players' actions

Extensive-Form Games (EFGs)

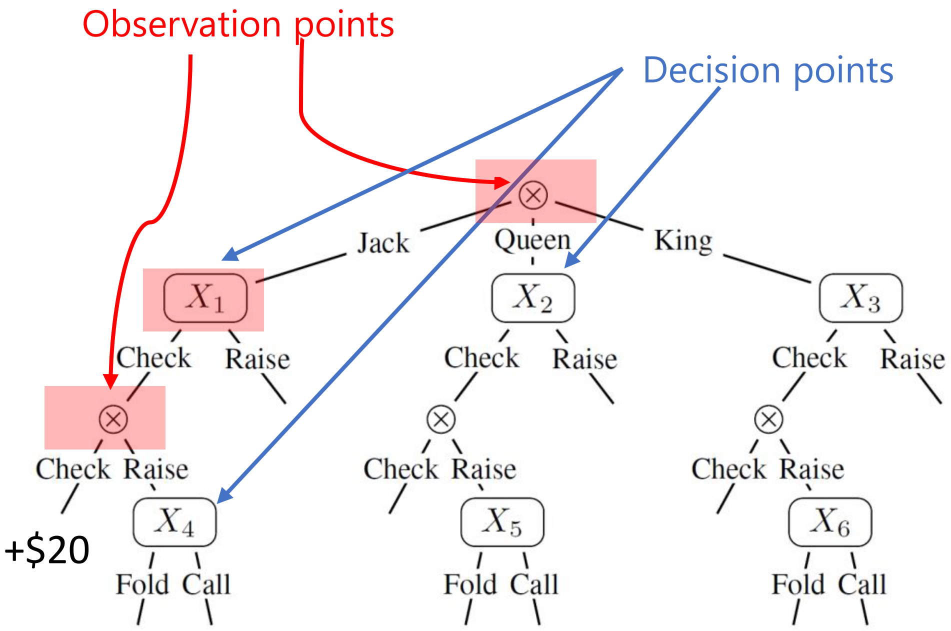
Each player faces a tree-form decision problem

EFGs capture both sequential and simultaneous moves, as well as imperfect information and stochastic moves

Very expressive model of interaction
Examples of EFGs: chess, poker, bridge, security games, …

Extensive-Form Games (EFGs)

Example: decision problem of Player 1 in ***Kuhn poker***



- **Decision points**:
  The decision maker picks one action from a set of available actions
- **Observation points**:
  The decision maker observes a signal drawn from a set of possible signals

Decision and observation points form a tree

Representing strategies in extensive-form games in a way that is optimization- and learning-friendly is not a priori 100% obvious

Good news: there exists a way of representing strategies in EFGs so that:

- *Each player's strategy set is a low-dimensional convex polytope ("sequence-form polytope")*
- *Utility functions are multilinear*

This enables online learning in extensive-form games, as well as other convex optimization techniques

Reality: online learning results for EFGs are harder to come by, due to their more intricate strategy sets

Normal-Form Games	***Extensive-Form Games***
• Per-player regret bound: • ☑ Polylog dependence on the number of actions • ☑ Polylog(T) dependence on time	✗ Not known
• Sum of players' regrets • ☑ Polylog dependence on #actions • ☑ Constant dependence on time	
• ☑ Last-strategy convergence*	▨ Less is known

For many years, the EFG community has been "chasing" the NFG community, extending NFG breakthroughs to EFGs, when possible

For example, all these were all developed later for EFGs than NFGs (and sometimes only with weaker guarantees):

- Good distance measures [Hoda et al. '10; Kroer et al. '15; Farina et al. '21]
- Efficient optimistic algorithms [Farina et al. '19]
- Last-iterate convergence [Wei et al. '21, Lee et al. '21]

In fact, this paper was born from our desire to extend the polylog(T) regret bounds by [Daskalakis et al. '21] to EFGs.

For many years, the EFG community has been "chasing" the NFG community, extending NFG breakthroughs to EFGs, when possible

For example, all these were all developed later for EFGs than NFGs (and sometimes only with weaker guarantees):

- Good distance measures [Hoda et al. '10; Kroer et al. '15; Farina et al. '21]
- Efficient optimistic algorithms [Farina et al. '19]
- Last-iterate convergence [Wei et al. '21, Lee et al. '21]

Does it have to be like that? Or can we somehow bridge the gap and inherit the best properties of NFG algorithms also in EFGs?

Can we somehow bridge the gap?

Folklore result: any EFG can be converted into an equivalent NFG where each player's action set is the set of all deterministic policies in their tree-form decision problem. So, if we applied OMWU to that....

Catch: the number of such policies is exponential in each player's tree size

Common wisdom: because of the exponential blowup, the above approach is a *computational dead end*

⚡ Consequence: specialized techniques were developed for EFGs, and progress on EFGs and NFGs follows separate tracks for decades

The common wisdom is wrong

This paper: It is possible to simulate OMWU on the normal-form equivalent of an EFGs, in ***linear time per iteration*** in the tree size, via a ***kernel trick***

We call our algorithm **Kernelized OMWU (KOMWU)**

In fact, kernelized OMWU applies to any polyhedral domain with 0/1-coordinate vertices $\Omega \subseteq \mathbb{R}^d$

Main theorem: OMWU on the set of vertices of Ω can be simulated using $d + 1$ evaluations of the kernel at each iteration

So, if each kernel evaluation can be performed in poly(d) time, OMWU can be simulated in poly(d) time

KOMWU **closes part of the gap** between learning in NFGs and EFGs

- It achieves all the strong properties of OMWU there were so far only known to be achievable efficiently in NFGs (including polylog regret)
- …as well as any future regret bounds that might get proven for OMWU!

As an unexpected byproduct, KOMWU obtains new state-of-the-art regret bounds among all online learning algorithms for extensive-form problems

Kernelized Multiplicative Weights for 0/1-Polyhedral Games

Algorithm		Per-player regret bound	Last-iter. conv.[†]
CFR (regret matching / regret matching$^+$)	(Zinkevich et al., 2007)	$\mathcal{O}(\sqrt{A}\,\lVert Q\rVert_1\,T^{1/2})$	no
CFR (MWU)	(Zinkevich et al., 2007)	$\mathcal{O}(\sqrt{\log A}\,\lVert Q\rVert_1\,T^{1/2})$	no
FTRL / OMD (dilated entropy)	(Kroer et al., 2020)	$\mathcal{O}(\sqrt{\log A}\,2^{D/2}\,\lVert Q\rVert_1\,T^{1/2})$	no
FTRL / OMD (dilatable global entropy)	(Farina et al., 2021a)	$\mathcal{O}(\sqrt{\log A}\,\lVert Q\rVert_1\,T^{1/2})$	no
Kernelized MWU	**(this paper)**	$\mathcal{O}(\sqrt{\log A}\,\sqrt{\lVert Q\rVert_1}\,T^{1/2})$	**no**
Optimistic FTRL / OMD (dilated entropy)	(Kroer et al., 2020)	$\mathcal{O}(\sqrt{m}\log(A)\,2^{D}\,\lVert Q\rVert_1^2\,T^{1/4})$	yes*
Optimistic FTRL / OMD (dilatable gl. ent.)	(Farina et al., 2021a)	$\mathcal{O}(\sqrt{m}\log(A)\,\lVert Q\rVert_1^2\,T^{1/4})$	no
Kernelized OMWU	**(this paper)**	$\mathcal{O}(m\log(A)\,\lVert Q\rVert_1\,\log^4(T))$	**yes**

Near-optimal O(polylog T) regret bound

Improved dependence on the ℓ_1 norm of the strategy space (half of the exponent)

Preliminaries

Online learning & normal-form games

Online Learning

Given a finite section of actions A, consider the following abstract model of a decision maker

- At each time t, the decision maker selects a distribution
$$\lambda^{(t)} \in \Delta(A) := \left\{\lambda \in \mathbb{R}^{A}_{\geq 0}: \sum_{a \in A} \lambda[a] = 1\right\}$$
- Then, the environment picks a reward vector $r^{(t)} \in \mathbb{R}_{\geq 0}$ and shows it to the decision maker
- Utility of decision maker is then the inner product $\langle \lambda^{(t)}, r^{(t)} \rangle$

Quality metric: regret $R^T := \max_{\hat{a} \in \Delta(A)} \sum_{t=1}^{T} \left\langle \hat{a}, r^{(t)} \right\rangle - \sum_{t=1}^{T} \left\langle \lambda^{(t)}, r^{(t)} \right\rangle$

Decision-making algorithms that ***guarantee sublinear regret in T in the worst case*** converge to equilibrium in games

Multiplicative weights update (MWU) is the most well-studied algorithm with that property

$\lambda^{(1)} := \frac{1}{|A|}\mathbf{1} \in \Delta(A)$

For $t = 1, 2, \ldots$

Output distribution $\lambda^{(t)}$

Observe reward vector $r^{(t)} \in \mathbb{R}^A$

Set $\lambda^{(t+1)}[a] := \frac{\lambda^{(t)}[a] \cdot e^{\eta\, r^{(t)}[a]}}{\sum_{a' \in A} \lambda^{(t)}[a'] \cdot e^{\eta\, r^{(t)}[a']}}$

Optimistic version obtained by replacing $r^{(t)}$ with $2r^{(t)} - r^{(t-1)}$

Normal-Form Games

- Simultaneous, nonsequential games
- Each player i picks an action from a finite set A_i, and received a payoff that depends on the combination of actions
- Strategy for each player: probability distribution λ_i over their actions A_i

Learning in games: each player repeatedly plays the game picking their distribution according to a learning algorithm

After each repetition, the reward vector of each agent is the gradient of the expected utility of that agent given the strategies of all other players

Polyhedral Convex Games

Polyhedral Convex Games

Idea: in a polyhedral convex game, the set of "strategies" of each player is given as a convex polytope $\Omega_i \subseteq \mathbb{R}^{d_i}$

$$\Gamma = (m, \{\Omega_i\}, \{\bar{U}_i\})$$

Number of players

Multilinear utility function for player i

$\bar{U}_i : \Omega_1 \times \cdots \times \Omega_m \to [0, 1]$

💡 the concepts of learning agent and equilibria directly extend to polyhedral games by replacing each $\Delta(A_i)$ with Ω_i

⚡ Extensive-form games are polyhedral convex games

Convex games: [Gordon et al. '08]

Polyhedral convex games can always be converted into an equivalent NFG in which each player i's action set is the set of vertices of Ω_i

This is what people mean when they talk about "the normal-form equivalent of an extensive-form game"

Change of variable: instead of picking a $x \in \Omega_i$, we instead pick convex combination coefficients $\lambda_i \in \Delta(V_i)$ over the vertices V_i of Ω_i

The process of learning in the normal-form equivalent using MWU can be written directly as MWU that tracks regret over the vertices

Vertex MWU algorithm

Setup
$\Omega_i \subseteq \mathbb{R}^d$
V_i vertices of Ω_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 $r^{(t)} \in \mathbb{R}^d$

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

The process of learning in the normal-form equivalent using MWU can be written directly as MWU that tracks regret over the vertices

Vertex MWU algorithm

Setup

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

V_i vertices of Ω_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 $r^{(t)} \in \mathbb{R}^d$

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

As usual, vertex **O**MWU is analogous

Vertex OMWU guarantees polylog T regret when used by all players

The process of learning in the normal-form equivalent using MWU can be written directly as MWU that **over the set of vertices**

Vertex MWU algorithm

Setup

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

V_i vertices of Ω_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 $r^{(t)} \in \mathbb{R}^d$

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

Main question of this paper:

Can Vertex (O)MWU be simulated efficiently?

Vertex MWU algorithm

Setup

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

V_i vertices of Ω_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 $r^{(t)} \in \mathbb{R}^d$

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

Main theorem

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

Crucially independent on the number of vertices of Ω_i!

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

The 0/1-Polyhedral Kernel

Setup

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

V vertices of Ω

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

Definition (0/1-feature map of Ω)

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

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 Ω)

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

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

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

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

Proof: by induction

Vertex MWU algorithm

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

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

For $t = 1, 2, \ldots$

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

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

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

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

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

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 Ω
$V \subseteq \{0,1\}^d$

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

For $t = 1, 2, \ldots$

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

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

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

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

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

Vertex MWU algorithm

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

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

For $t = 1, 2, \ldots$

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

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

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

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)

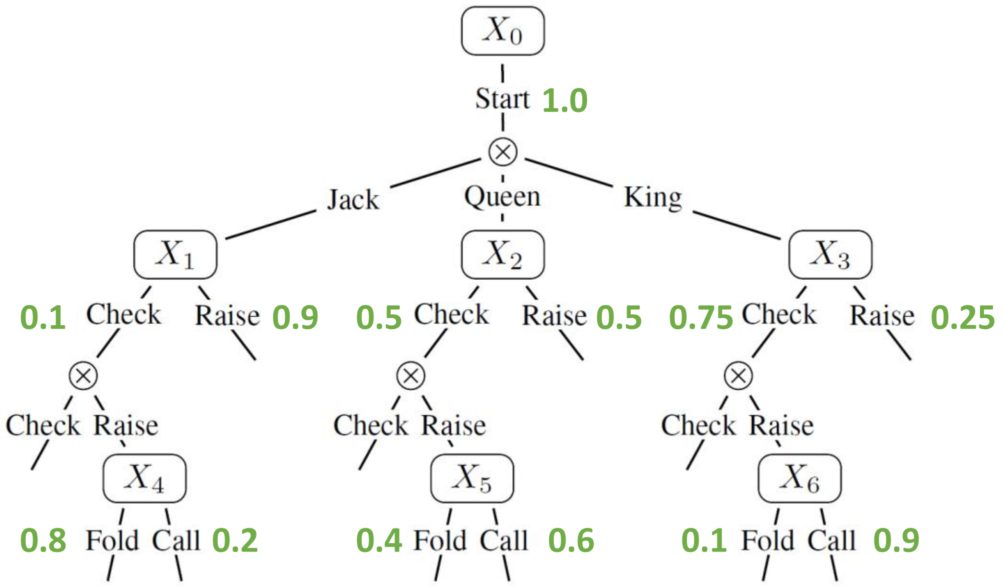
Proof: extends a nice and simple insight of Takimoto and Warmuth

Vertex MWU algorithm	**Kernelized MWU** algorithm
Setup $\Omega \subseteq \mathbb{R}^d$ V vertices of Ω $V \subseteq \{0,1\}^d$	***Setup*** $\Omega \subseteq \mathbb{R}^d$ V vertices of Ω $V \subseteq \{0,1\}^d$
$\lambda^{(1)} := \frac{1}{\lvert V \rvert}\mathbf{1} \in \mathbb{R}^V$	$b^{(1)} := \mathbf{1} \in \mathbb{R}^d$
For $t = 1, 2, \ldots$	For $t = 1, 2, \ldots$
Play $x^{(t)} := \sum_{v \in V_i} \lambda^{(t)}[v] \cdot v$	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 reward $r^{(t)} \in \mathbb{R}^d$	Observe reward $r^{(t)} \in \mathbb{R}^d$
Set $\lambda^{(t+1)}[v] := \frac{\lambda^{(t)}[v] \cdot e^{\eta \langle r^{(t)}, v \rangle}}{\sum_{v' \in V} \lambda^{(t)}[v'] \cdot e^{\eta \langle r^{(t)}, v' \rangle}}$	Set $b^{(t+1)} := \exp\{\eta \sum_{\tau=1}^{t} r^{(\tau)}\}$

Kernel in Extensive-Form Games

In order to see an intuition for how to evaluate the kernel in extensive-form games, it is important to *understand the geometry of the sequence-form strategy sets* Ω_i

Strategies in Extensive-Form Games



``Behavioral strategies''

First attempt:

Assign local probabilities at each decision point

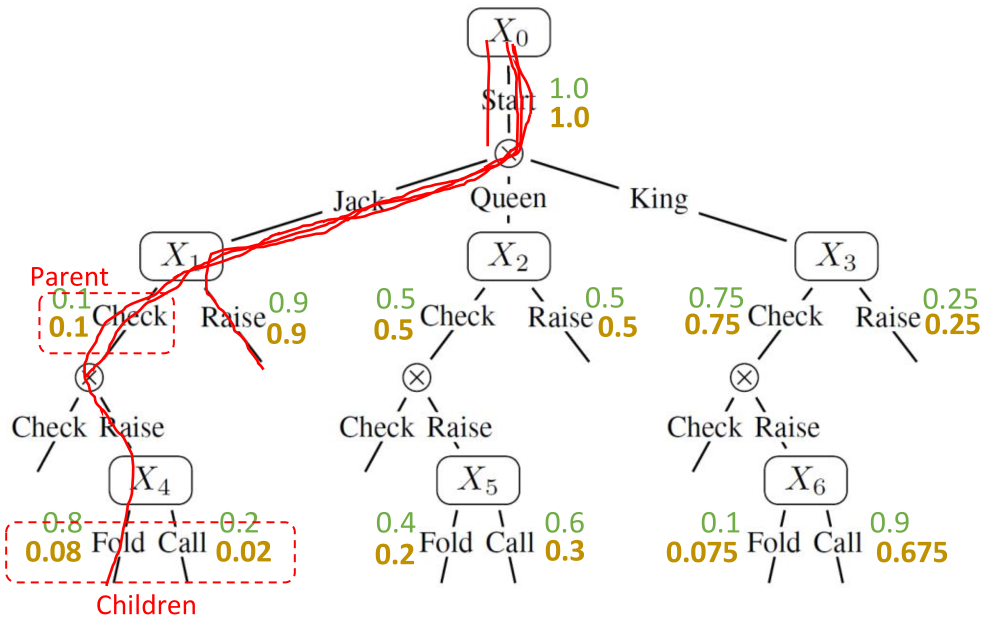
✔ Set of strategies is convex

✘ Expected utility is **not** linear

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 is a linear function

Consistency constraints

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

Convex polytope Ω_i

[Romanovskii, Reduction of a game with complete memory to a matrix game, 1962]
[Koller et al., Fast algorithms for finding randomized strategies in game trees, STOC 1994]

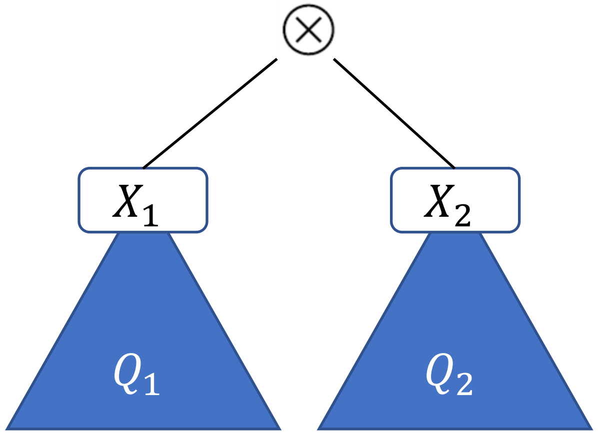
Kernel of Ω_i

Theorem: given any x, y we can evaluate the kernel $K_{\Omega_i}(x, y)$ in time linear in the number of edges of the tree-form decision problem

Corollary: we can implement KOMWU with *quadratic* time per iteration in the decision tree size

Intuition

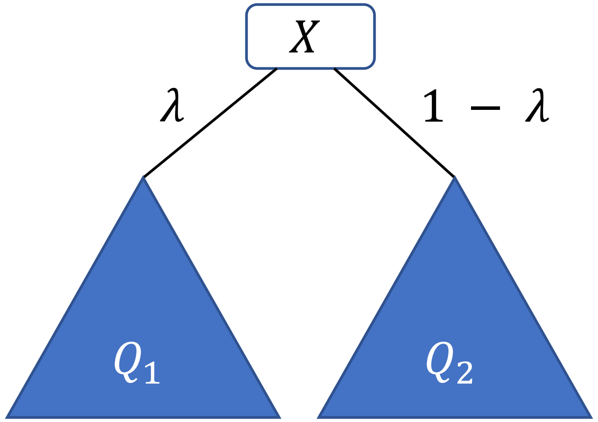
Idea: Sequence-form strategy spaces have a strong bottom up combinatorial structure!



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

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

Convex Hulls

Intuition

- We exploit the combinatorial structure by introducing “partial kernels” for subtrees of the tree-form decision problem

- At every decision point X, the kernel for the subtree rooted at X is a weighted **sum** of the kernels rooted in each of the child subtrees
- At every observation point Y, the kernel for the subtree rooted at Y is the **product** of the kernels rooted in each of the child subtrees

- This gives a linear-time bottom-up computation of the kernel

Kernel of Ω_i

Theorem: given any x, y we can evaluate the kernel $K_{\Omega_i}(x, y)$ in time linear in the number of edges of the tree-form decision problem

Corollary: we can implement KOMWU with *quadratic* time per iteration in the decision tree size

Kernel of Ω_i

Can we do better than quadratic iterations?

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

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

Can we amortize the cost of computing those d + 1 kernels?

Kernel of Ω_i

Corollary: we can implement KOMWU with *linear* time per iteration in the decision tree size, by amortizing the complexity of the d+1 kernel evaluation by reusing intermediate computations

In summary, in extensive-form games KOMWU guarantees:

- Linear-time iterations
- Polylog regret when used by all players in the EFG (*for the first time*)
- More favorable regret bounds than all prior known EFG algorithms
- *Future proof:* if the analysis of OMWU's regret is further improved for NFGs, the improvement will propagate to EFGs

Summary and Open Questions

Summary

- We introduced Kernelized OMWU
- It simulates running OMWU on the vertices of a 0/1-polyhedral set via black-box access to a kernel function
- The kernel function can be evaluated in linear time in the size of the tree-form decision problem in extensive-form games
- It defies a long held common wisdom about extensive-form games…
- … and leads to new state-of-the-art regret bounds for EFGs

Other sets for which the kernel can be evaluated efficiently

- Unit hypercube $\Omega = [0,1]^n$

$$K_\Omega(x, y) = (1 + x_1 y_1) \cdots (1 + x_n y_n)$$

- Set of flows in a DAG (dynamic programming on topological ordering)
- Doubly stochastic matrices (only approximate computation)
- N-sets: $\mathrm{co}\{x \in \{0,1\}^d : \ ||x||_1 = n\}$
    - Dynamic programming
- Spanning trees
- In many cases, KOMWU unifies existing approaches for particular combinatorial sets under a unified framework

Inspirations

- We are especially indebted to the work by Takimoto and Warmuth on path kernels for graphs for some of the precursor work
- The kernel used by KOMWU can be seen as a significant generalization of Takimoto and Warmuth's *path kernel* for DAGs

Open Questions

What can be said beyond 0/1-coordinate vertices? Can we somehow develop a more advanced kernel function?

Can near-optimal regret bounds be guaranteed for ***general*** convex games?

Thanks!