

---

# Kernelized Multiplicative Weights for 0/1-Polyhedral Games: Bridging the Gap Between Learning in Extensive-Form and Normal-Form Games

---

Gabriele Farina<sup>1</sup> Chung-Wei Lee<sup>2</sup> Haipeng Luo<sup>2</sup> Christian Kroer<sup>3</sup>

## Abstract

While extensive-form games (EFGs) can be converted into normal-form games (NFGs), doing so comes at the cost of an exponential blowup of the strategy space. So, progress on NFGs and EFGs has historically followed separate tracks, with the EFG community often having to catch up with advances (*e.g.*, last-iterate convergence and predictive regret bounds) from the larger NFG community. In this paper we show that the Optimistic Multiplicative Weights Update (OMWU) algorithm—the premier learning algorithm for NFGs—can be simulated on the normal-form equivalent of an EFG in linear time per iteration in the game tree size using a kernel trick. The resulting algorithm, *Kernelized OMWU (KOMWU)*, applies more broadly to all convex games whose strategy space is a polytope with 0/1 integral vertices, as long as the kernel can be evaluated efficiently. In the particular case of EFGs, KOMWU closes several standing gaps between NFG and EFG learning, by enabling direct, black-box transfer to EFGs of desirable properties of learning dynamics that were so far known to be achievable only in NFGs. Specifically, KOMWU gives the first algorithm that guarantees at the same time last-iterate convergence, lower dependence on the size of the game tree than all prior algorithms, and  $\tilde{O}(1)$  regret when followed by all players.

rounds of repeated play using a no-regret learning algorithm such as multiplicative weights update (MWU), the average product distribution of play is a  $\mathcal{O}(1/\sqrt{T})$ -approximate Nash equilibrium in two-player zero-sum games, and a  $\mathcal{O}(1/\sqrt{T})$ -approximate coarse-correlated equilibrium in multiplayer general-sum games. In the last decade, much stronger results have been obtained when each player employs an *optimistic* no-regret learner such as the *optimistic MWU* (OMWU) algorithm (Rakhlin & Sridharan, 2013a,b; Syrgkanis et al., 2015). For example, in zero-sum NFGs OMWU enables convergence to a Nash equilibrium at a rate of  $\mathcal{O}(1/T)$  and various *last-iterate* guarantees (Daskalakis & Panageas, 2019; Lei et al., 2021; Wei et al., 2021). For general-sum NFGs, polylogarithmic regret bounds have been shown when every player uses OMWU (Daskalakis et al., 2021), implying convergence to a coarse-correlated equilibrium at a  $\tilde{O}(1/T)$  rate.

In this paper we study *extensive-form games* (EFGs), a much richer class of games that explicitly model sequential (or simultaneous) interaction, stochastic outcomes, and imperfect information. Because of their sequential nature, the number of deterministic strategies in an EFG is exponential in the size of the game, unlike for NFGs. Computing, or approximating, Nash equilibria of large EFGs has been a key component of recent AI milestones where AIs were created that beat human poker players (Bowling et al., 2015; Brown & Sandholm, 2019; 2017; Moravčík et al., 2017). These results relied on online learning algorithms for the decision sets of the players in an EFG, where each iteration of the algorithm is performed in linear time in the game tree size (which is crucial due to the large size of these games).

Online learning results for EFGs are generally somewhat harder to come by, and have often lagged behind results for NFGs. This is due to the more complicated combinatorial structure of the decision spaces in EFGs. For example, the following concepts were all developed later for EFGs than for NFGs, and sometimes with weaker guarantees: good distance measures (Hoda et al., 2010; Kroer et al., 2015; 2020; Farina et al., 2021a), optimistic regret-minimization algorithms (Farina et al., 2019b;a), and last-iterate convergence results (Wei et al., 2021; Lee et al., 2021). Very recent NFG results such as the polylogarithmic regret bounds for

## 1. Introduction

Online learning in the context of *normal-form games* (NFGs) has been studied extensively. A classic motivation for this study is that when every player in an NFG learns from  $T$

<sup>1</sup>Computer Science Department, Carnegie Mellon University <sup>2</sup>Computer Science Department, University of Southern California <sup>3</sup>IEOR Department, Columbia University. Correspondence to: Gabriele Farina <gfarina@cs.cmu.edu>, Chung-Wei Lee <leechung@usc.edu>, Haipeng Luo <haipengl@usc.edu>, Christian Kroer <christian.kroer@columbia.edu>.

**Kernelized Multiplicative Weights for 0/1-Polyhedral Games**

Algorithm		Per-player regret bound	Last-iter. conv. <sup>†</sup>
CFR (regret matching / regret matching <sup>+</sup> )	(Zinkevich et al., 2007)	$\mathcal{O}(\sqrt{A} \ Q\ _1 T^{1/2})$	no
CFR (MWU)	(Zinkevich et al., 2007)	$\mathcal{O}(\sqrt{\log A} \ Q\ _1 T^{1/2})$	no
FTRL / OMD (dilated entropy)	(Kroer et al., 2020)	$\mathcal{O}(\sqrt{\log A} 2^{D/2} \ Q\ _1 T^{1/2})$	no
FTRL / OMD (dilatable global entropy)	(Farina et al., 2021a)	$\mathcal{O}(\sqrt{\log A} \ Q\ _1 T^{1/2})$	no
<b>Kernelized MWU</b>	<b>(this paper)</b>	$\mathcal{O}(\sqrt{\log A} \sqrt{\ Q\ _1} T^{1/2})$	<b>no</b>
Optimistic FTRL / OMD (dilated entropy)	(Kroer et al., 2020)	$\mathcal{O}(\sqrt{m} \log(A) 2^D \ Q\ _1^2 T^{1/4})$	yes*
Optimistic FTRL / OMD (dilatable gl. ent.)	(Farina et al., 2021a)	$\mathcal{O}(\sqrt{m} \log(A) \ Q\ _1^2 T^{1/4})$	no
<b>Kernelized OMWU</b>	<b>(this paper)</b>	$\mathcal{O}(m \log(A) \ Q\ _1 \log^4(T))$	<b>yes</b>

Table 1. Properties of various no-regret algorithms for EFGs. All algorithms take linear time to perform an iteration. The first set of rows are for non-optimistic algorithms. The second set of rows are for optimistic algorithms. The regret bounds are per player and apply to multiplayer general-sum games. They depend on the maximum number of actions  $A$  available at any decision point, the maximum  $\ell_1$  norm  $\|Q\|_1 = \max_{q \in Q} \|q\|_1$  over the player’s decision polytope  $Q$ , the depth  $D$  of the decision polytope, and the number of players  $m$ . Optimistic algorithms have better asymptotic regret, but worse dependence on the game constants  $m$ ,  $A$ , and  $\|Q\|_1$ . Note that our algorithms achieve better dependence on  $\|Q\|_1$  compared to all existing algorithms. <sup>†</sup>Last-iterate convergence results are for two-player zero-sum games, and some results rely on the assumption of a unique Nash equilibrium—see Section 5.3 for details. \*Lee et al. (2021).

OMWU dynamics in general-sum NFGs (Daskalakis et al., 2021) do not currently have an analogue for EFGs.

In principle, an EFG can be represented as a NFG where each action in the NFG corresponds to an assignment of decisions at *each* decision point in the EFG. One could then run, *e.g.*, OMWU on this normal-form representation, and receive all the guarantees obtained for NFGs directly. However, this reduction is exponentially-large in the size of the EFG representation, and for this reason the normal-form representation was viewed as impractical. This leads to the necessity of developing the various more complicated approaches mentioned in the previous paragraph.

We contradict popular belief and show that it is possible to work with the normal form efficiently: we provide a kernel-based reduction from EFGs to NFGs that allows us to simulate MWU and OMWU on the normal-form representation, using only linear (in the EFG size) time per iteration. Our algorithm, *Kernelized OMWU* (KOMWU), closes the gap between NFGs and EFGs; KOMWU achieves all the guarantees provided by the various normal-form results mentioned previously, as well as any future results on OMWU for NFGs. As an unexpected byproduct, KOMWU obtains new state-of-the-art regret bounds among all on-line learning algorithms for EFGs (see also Table 1); we improve the dependence on the maximum  $\ell_1$  norm  $\|Q\|_1$  over the sequence-form polytope  $Q$  from  $\|Q\|_1^2$  to  $\|Q\|_1$  (for the non-optimistic version we improve it from  $\|Q\|_1$  to  $\sqrt{\|Q\|_1}$ ). Due to the connection between regret minimization and convergence to Nash equilibrium, this also improves the state-of-the-art bounds for converging to a Nash equilibrium at either a rate of  $1/\sqrt{T}$  or  $1/T$  by the same factor. Moreover, KOMWU achieves last-iterate convergence, and as it is the first algorithm to achieve linear-rate last-iterate convergence with a learning rate that does not

become impractically-small as the game grows large (albeit under a restrictive uniqueness assumption).

More generally, we show that KOMWU can simulate OMWU for *0/1-polyhedral sets* (of which the decision sets for EFGs are a special case): a decision set  $\Omega \subseteq \mathbb{R}^d$  which is convex and polyhedral, and whose vertices are all contained in  $\{0, 1\}^d$ . KOMWU reduces the problem of running OMWU on the vertices of the polyhedral set to  $d + 1$  evaluations of what we call the *0/1-polyhedral kernel*. Thus, given an efficient algorithm for performing these kernel evaluations, KOMWU enables one to get all the benefits of running MWU or OMWU on the simplex of vertices, while retaining the crucial property that each iteration of OMWU can be performed efficiently. In addition to EFGs, in the appendix we show that the kernel can be computed efficiently for several other settings including  $n$ -sets, unit cubes, flows on directed acyclic graphs, and permutations. As with EFGs, this immediately gives us an efficient algorithm with favorable properties such as last-iterate convergence and polylogarithmic regret for games with 0/1-polyhedral strategy sets. In particular, for  $n$ -sets, we show an improvement on the time complexity per round compared with the dynamic programming approach discussed in (Takimoto & Warmuth, 2003). To the best of our knowledge, this is the state-of-the-art bound for simulating MWU/OMWU on  $n$ -sets.

**Related work** There were several past works on specialized online learning methods for EFGs. One class of methods is based on specialized Bregman divergences that lead to efficient iteration updates (Hoda et al., 2010; Kroer et al., 2015; 2020; Farina et al., 2021a). Combined with optimistic regret minimizers for general convex set, this yields stronger regret bounds that take into account the variation in payoffs, and combined with the connection between regret minimization and Nash equilibrium computation, this yields  $1/T$ -

rate convergence for two-player zero-sum games (Rakhlin & Sridharan, 2013b; Syrgkanis et al., 2015; Farina et al., 2019b). The *counterfactual regret minimization* (CFR) framework Zinkevich et al. (2007) also yields efficient iteration updates. This approach yields a worse  $\sqrt{T}$  regret bound, but leads to the best practical performance in most games (Kroer et al., 2018; 2020; Farina et al., 2021b). Farina et al. (2019a) show that it is possible to attain  $\mathcal{O}(T^{1/4})$  regret within the CFR framework by using OMWU at each decision point. However, the game-dependent constants in their bound are much worse than the ones in Table 1.

Regret minimization over 0/1 polyhedral sets, the framework we consider, is closely related to online combinatorial optimization problems (Audibert et al., 2014), where the decision maker (randomly) selects a 0/1 vertex in each round instead of a point in the convex hull of the set of vertices, and the regret is measured in expectation. We review approaches related to the use of MWU here, and other less closely related approaches in Appendix A. One approach similar to our KOMWU is to perform MWU over vertices (e.g., Cesa-Bianchi & Lugosi (2012)); the remaining problem is whether there is an efficient way to maintain and sample from the weights. Such efficient implementations have been shown in many instances such as paths (Takimoto & Warmuth, 2003), spanning trees (Koo et al., 2007), and  $m$ -set (Warmuth & Kuzmin, 2008). (Takimoto & Warmuth, 2003) is the closest to this paper, where they show how to produce MWU iterates for paths in directed graphs. Our kernelized method can be seen as a significant extension of their approach to general 0/1 polyhedral games, unifying many of the previous results listed above. This unification not only results in important applications to EFGs, but also leads to improvement to previously studied problems such as  $n$ -sets.

## 2. Preliminaries

In this section we review some fundamental connections between normal-form games and no-regret learners.

### 2.1. Online Learning and Multiplicative Weights Update

Given a finite set of choices  $\mathcal{A}$ , consider the following abstract model of a repeated decision-making problem between a decision maker and an unknown—potentially adversarial—environment. At each time  $t = 1, 2, \dots$ , the decision maker is given (or otherwise selects) a *prediction vector*  $\mathbf{m}^{(t)} \in \mathbb{R}^{\mathcal{A}}$ . Then, the decision maker must select and output a probability distribution  $\boldsymbol{\lambda}^{(t)}$  over  $\mathcal{A}$ , that is, a vector  $\boldsymbol{\lambda}^{(t)} \in \Delta(\mathcal{A}) := \{\boldsymbol{\lambda} \in \mathbb{R}_{\geq 0}^{\mathcal{A}} : \sum_{a \in \mathcal{A}} \lambda[a] = 1\}$ . Finally, the environment picks (possibly in an adversarial way) a *loss vector*  $\boldsymbol{\ell}^{(t)} \in \mathbb{R}^{\mathcal{A}}$  and shows it to the decision maker, who then suffers a loss equal to  $\langle \boldsymbol{\ell}^{(t)}, \boldsymbol{\lambda}^{(t)} \rangle$ . Given

any time  $T$ , a key quantity for the decision maker is its *cumulative regret* (or simply *regret*) up to time  $T$ ,

$$R^T := \sum_{t=1}^T \langle \boldsymbol{\ell}^{(t)}, \boldsymbol{\lambda}^{(t)} \rangle - \min_{\hat{\boldsymbol{\lambda}} \in \Delta(\mathcal{A})} \sum_{t=1}^T \langle \boldsymbol{\ell}^{(t)}, \hat{\boldsymbol{\lambda}} \rangle. \quad (1)$$

As we recall in the next subsection, decision-making algorithms that guarantee sublinear regret (in  $T$ ) in the worst case make for natural agents to learn equilibria in games. The most well-studied decision-making algorithm with that property is the *optimistic multiplicative weights update* (OMWU) algorithm.<sup>1</sup> Let  $\boldsymbol{\ell}^{(0)}, \mathbf{m}^{(0)} := \mathbf{0} \in \mathbb{R}^{\mathcal{A}}$  and  $\boldsymbol{\lambda}^{(0)} := \frac{1}{|\mathcal{A}|} \mathbf{1} \in \Delta(\mathcal{A})$ ; then, at all times  $t \in \mathbb{N}_{>0}$ , OMWU updates the distribution  $\boldsymbol{\lambda}^{(t-1)} \in \Delta(\mathcal{A})$  according to

$$\boldsymbol{\lambda}^{(t)}[a] := \frac{\boldsymbol{\lambda}^{(t-1)}[a] \cdot e^{-\eta^{(t)} \mathbf{w}^{(t)}[a]}}{\sum_{a' \in \mathcal{A}} \boldsymbol{\lambda}^{(t-1)}[a'] \cdot e^{-\eta^{(t)} \mathbf{w}^{(t)}[a']}} \quad (\blacklozenge)$$

for all  $a \in \mathcal{A}$ , where  $\mathbf{w}^{(t)} := \boldsymbol{\ell}^{(t-1)} - \mathbf{m}^{(t-1)} + \mathbf{m}^{(t)}$  and  $\eta^{(t)} > 0$  is a learning rate (full pseudocode is given in Appendix B). The nonpredictive version of OMWU, called *multiplicative weights update* (MWU), is obtained from OMWU as the special case in which  $\mathbf{m}^{(t)} = \mathbf{0}$  at all  $t$ .

### 2.2. Normal-Form Games (NFGs)

*Normal-form games* (NFG) are simultaneous-move, nonsequential games in which each player picks an action from a finite set, and receives a payoff that depends on the tuple of actions played by the players. Formally, we represent a normal form game as a tuple  $\Gamma = (m, \{\mathcal{A}_i\}, \{U_i\})$ , where the positive integer  $m \in \mathbb{N}_{>0}$  denotes the number of players, each of which is assigned a unique player number in the set  $\llbracket m \rrbracket := \{1, \dots, m\}$ ; the finite set  $\mathcal{A}_i$  specifies the actions available to player  $i \in \llbracket m \rrbracket$ ; and  $U_i : \mathcal{A}_1 \times \dots \times \mathcal{A}_m \rightarrow [0, 1]$  is the payoff function for player  $i \in \llbracket m \rrbracket$ . The game is said to be *zero-sum* if  $\sum_{i \in \llbracket m \rrbracket} U_i(a_1, \dots, a_m) = 0$  for all  $(a_1, \dots, a_m) \in \mathcal{A}_1 \times \dots \times \mathcal{A}_m$ .

A *mixed strategy* for any player  $i \in \llbracket m \rrbracket$  is a probability distribution  $\boldsymbol{\lambda}_i \in \Delta(\mathcal{A}_i)$  over the player’s action set  $\mathcal{A}_i$ . When the players play according to mixed strategies  $\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_m$ , the *expected utility*  $\bar{U}_i$  of any player  $i \in \llbracket m \rrbracket$  is defined accordingly as the function  $\bar{U}_i : (\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_m) \mapsto \mathbb{E}_{a_1 \sim \boldsymbol{\lambda}_1, \dots, a_m \sim \boldsymbol{\lambda}_m} [U_i(a_1, \dots, a_m)]$ . Because of the linearity of expectation, the expected utility function  $\bar{U}_i$  of each player  $i$  is a *multilinear* function of the strategies  $\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_m$ .

**Learning in NFGs** We now describe a learning setup for NFGs, which we will refer to as the *canonical optimistic learning setup* (COLS). In the COLS, the NFG is played repeatedly. At each time  $t \in \mathbb{N}_{>0}$ , each player  $i \in \llbracket m \rrbracket$

<sup>1</sup>In the literature, OMWU is often given under the assumption that  $\mathbf{m}^{(t)} = \boldsymbol{\ell}^{(t-1)}$  at all times  $t$ . In this paper we present OMWU in its general form, that is, with no assumptions on  $\mathbf{m}^{(t)}$ .

picks mixed strategies  $\lambda_i^{(t)} \in \Delta(\mathcal{A}_i)$  according to a learning algorithm  $\mathcal{R}_i$ , with the following choice of loss and prediction vectors:

- The loss vector  $\ell_i^{(t)}$  is the opposite of the gradient of the expected utility of player  $i$  with respect of player  $i$ 's strategy, in symbols  $\ell_i^{(t)} := -\nabla_{\lambda_i} \bar{U}_i(\lambda_1^{(t)}, \dots, \lambda_m^{(t)})$ ;
- The prediction vector  $m_i^{(t)}$  is defined as the previous loss  $m_i^{(t)} := \ell_i^{(t-1)}$  if  $t \geq 2$ , and  $m_i^{(1)} := \mathbf{0}$  otherwise.

This is the same setup that was used in landmark papers such as (Syrkkanis et al., 2015) and (Daskalakis et al., 2021). A key result in the theory of learning in games establishes a deep connection between the COLS and *coarse-correlated equilibria* (CCEs) of the game (which, in two-player zero-sum games, are Nash equilibria).

**Theorem 2.1.** *Under the COLS, the average product distribution of play  $\bar{\mu} := \frac{1}{T} \sum_{t=1}^T \lambda_1^{(t)} \otimes \dots \otimes \lambda_m^{(t)}$  is an  $\mathcal{O}(\max_{i \in [m]} R_i^T / T)$ -approximate CCE of the game, where  $R_i^T$  is the regret for player  $i$  (see Eq. (1)).*

When each player  $i$  learns under the COLS using OMWU with the same, constant learning rate  $\eta_i^{(t)} := \eta$  as their learning algorithm  $\mathcal{R}_i$ , the following strong properties hold for any NFG  $\Gamma = (m, \{\mathcal{A}_i\}, \{U_i\})$ .

**Property 2.2** (Near-optimal per-player regret). *There exist universal constants  $C, C' > 1$  so that, for all  $T$ , if  $\eta \leq \frac{1}{C_m \log^4 T}$ , the regret of each player  $i \in [m]$  is bounded as  $R_i^T \leq \frac{\log |\mathcal{A}_i|}{\eta} + C' \log T$  (Daskalakis et al., 2021).*

**Property 2.3** (Optimal regret sum). *If  $\eta \leq \frac{1}{\sqrt{8(m-1)}}$ , at all times  $T \in \mathbb{N}_{>0}$  the sum of the players' regrets satisfies  $\sum_{i=1}^m R_i^T \leq \frac{m}{\eta} \max_{i=1}^m \log |\mathcal{A}_i|$  (Syrkkanis et al., 2015).*

When  $\Gamma$  is a *two-player zero-sum* game, the following also holds when learning under the COLS using OMWU.

**Property 2.4** (Last-iterate convergence). *There exists a certain schedule of learning rates  $\eta_i^{(t)}$  such that the players' strategies  $(\lambda_1^{(t)}, \lambda_2^{(t)})$  converge to a Nash equilibrium of the game (Hsieh et al., 2021). Furthermore, if  $\Gamma$  has a unique Nash equilibrium  $(\lambda_1^*, \lambda_2^*)$  and each player uses any constant learning rate  $\eta_i^{(t)} := \eta \leq \frac{1}{8}$ , at all times  $t$  the strategy profile  $(\lambda_1^{(t)}, \lambda_2^{(t)})$  satisfies  $D_{\text{KL}}(\lambda_1^* \parallel \lambda_1^{(t)}) + D_{\text{KL}}(\lambda_2^* \parallel \lambda_2^{(t)}) \leq C(1+C')^{-t}$ , where the constants  $C, C'$  only depend on the game, and  $D_{\text{KL}}(\cdot \parallel \cdot)$  denotes the KL-divergence between two distributions (Wei et al., 2021).*

### 3. Multiplicative Weights in Polyhedral Convex Games

A powerful generalization of normal-form games is *polyhedral convex games*, of which extensive-form games are an example (Gordon et al., 2008). Unlike NFGs, in which

players select a mixed strategy from the probability simplex spanned by the set of available action  $\mathcal{A}_i$ , in a polyhedral convex game the set of “randomized strategies” from which each player  $i \in [m]$  can draw is a given convex polytope  $\Omega_i \subseteq \mathbb{R}^{d_i}$ . Analogously to NFGs, we represent a polyhedral convex game as a tuple  $\Gamma = (m, \{\Omega_i\}, \{\bar{U}_i\})$ , where the functions  $\bar{U}_i : \Omega_1 \times \dots \times \Omega_m \rightarrow [0, 1]$  are the *multilinear* utility functions for each player  $i \in [m]$ .

The concepts of learning agents, equilibria, and COLS introduced in Sections 2.1 and 2.2 can be directly extended to polyhedral convex games without difficulty, by simply replacing the set of mixed strategies  $\Delta(\mathcal{A}_i)$  of each player with their convex polyhedral counterpart  $\Omega_i$ .

Because the set of mixed strategies  $\Omega$  of every player is a polytope, the decision problem of picking a mixed strategy  $x^{(t)} \in \Omega$  can be equivalently thought of as the decision problem of picking a convex combination  $\lambda^{(t)} \in \Delta(\mathcal{V}_\Omega)$  over the finite set of vertices  $\mathcal{V}_\Omega$  of  $\Omega$ . Indeed, it is not hard to show that a learning algorithm  $\mathcal{R}$  for  $\Omega \subseteq \mathbb{R}^d$  can be constructed from *any* learning algorithm  $\tilde{\mathcal{R}}$  for the set of vertices  $\mathcal{V}_\Omega$ , as we describe next. Let  $\mathbf{V}$  denote the matrix whose columns are the vertices  $\mathcal{V}_\Omega$ ; then:

- whenever  $\mathcal{R}$  receives a prediction  $m^{(t)} \in \mathbb{R}^d$  (resp., loss  $\ell^{(t)}$ ), it computes the vector  $\tilde{m}^{(t)} := \mathbf{V}^\top m^{(t)} \in \mathbb{R}^{|\mathcal{V}_\Omega|}$  (resp.,  $\tilde{\ell}^{(t)} := \mathbf{V}^\top \ell^{(t)}$ ) and forwards it to  $\tilde{\mathcal{R}}$ ;
- whenever  $\tilde{\mathcal{R}}$  plays a new distribution  $\lambda^{(t)} \in \Delta(\mathcal{V}_\Omega)$ , the convex combination of vertices  $x^{(t)} := \sum_{v \in \mathcal{V}_\Omega} \lambda^{(t)}[v] v = \mathbf{V} \lambda^{(t)}$  is played by  $\mathcal{R}$ .

It is immediate to verify that the regret cumulated by  $\mathcal{R}$  and  $\tilde{\mathcal{R}}$  is equal at all times  $T$ . So, as long as  $\tilde{\mathcal{R}}$  guarantees sublinear regret, then so does  $\mathcal{R}$ . In this paper we are particularly interested in the algorithm obtained by using the above construction for the specific choice of OMWU as the algorithm  $\tilde{\mathcal{R}}$ . We coin *Vertex OMWU* the resulting learning algorithm  $\mathcal{R}$  in that case, depicted in Figure 1. Let  $\ell^{(0)}, m^{(0)} := \mathbf{0} \in \mathbb{R}^{|\mathcal{V}_\Omega|}$  and  $\lambda^{(0)} := \frac{1}{|\mathcal{V}_\Omega|} \mathbf{1} \in \Delta(\mathcal{V}_\Omega)$ ; then, at all times  $t \in \mathbb{N}_{>0}$ , Vertex OMWU updates the convex combination of vertices  $\lambda^{(t-1)} \in \Delta(\mathcal{V}_\Omega)$  according to

$$\lambda^{(t)}[v] := \frac{\lambda^{(t-1)}[v] \cdot e^{-\eta^{(t)} \langle w^{(t)}, v \rangle}}{\sum_{v' \in \mathcal{V}_\Omega} \lambda^{(t-1)}[v'] \cdot e^{-\eta^{(t)} \langle w^{(t)}, v' \rangle}}, \quad (\clubsuit)$$

where

$$w^{(t)} := \ell^{(t-1)} - m^{(t-1)} + m^{(t)} \in \mathbb{R}^d, \quad (2)$$

and then outputs the iterate

$$\Omega \ni x^{(t)} := \sum_{v \in \mathcal{V}_\Omega} \lambda^{(t)}[v] \cdot v = \mathbf{V} \lambda^{(t)}. \quad (\spadesuit)$$

It is straightforward to show that Vertex OMWU satisfies Properties 2.2 to 2.4 with  $|\mathcal{A}_i|$  replaced with  $|\mathcal{V}_{\Omega_i}|$ ,

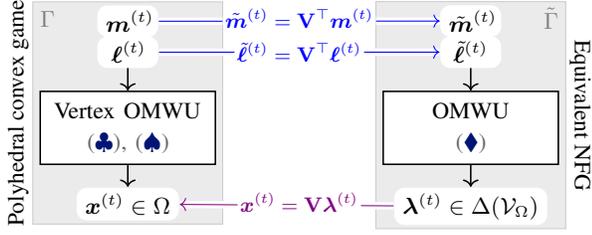


Figure 1. Construction of the Vertex OMWU algorithm. The matrix  $\mathbf{V}$  has the (possibly exponentially-many) vertices  $\mathcal{V}_\Omega$  of the convex polytope  $\Omega$  as columns.

by using a black-box reduction to NFGs. Indeed, let  $\Gamma = (m, \{\Omega_i\}, \{\bar{U}_i\})$  be a polyhedral convex game, and introduce the *NFG  $\tilde{\Gamma}$  equivalent to  $\Gamma$* , defined as the NFG  $\tilde{\Gamma} := (m, \{\mathcal{V}_{\Omega_i}\}, \{U_i\})$  where the action set of each player is the set of vertices  $\mathcal{V}_{\Omega_i}$ , and  $U_i(\mathbf{v}_1, \dots, \mathbf{v}_m) := \bar{U}_i(\mathbf{v}_1, \dots, \mathbf{v}_m)$  for all  $(\mathbf{v}_1, \dots, \mathbf{v}_m) \in \mathcal{V}_{\Omega_1} \times \dots \times \mathcal{V}_{\Omega_m}$ . Consider the losses  $\ell_i^{(t)}$ , predictions  $\mathbf{m}^{(t)}$ , and iterates  $\mathbf{x}_i^{(t)} \in \Omega_i$  produced by agents learning (under the COLS) in  $\Gamma$  using Vertex OMWU, and the losses  $\tilde{\ell}_i^{(t)}$ , predictions  $\tilde{\mathbf{m}}_i^{(t)}$ , and iterates  $\lambda_i^{(t)} \in \Delta(\mathcal{V}_i)$  produced by agents learning (again under the COLS) in  $\tilde{\Gamma}$  using OMWU. For all players  $i \in [m]$ , it is immediate to verify by induction that the relationships (i)  $\tilde{\ell}_i^{(t)} = \mathbf{V}_i^\top \ell_i^{(t)}$ , (ii)  $\tilde{\mathbf{m}}_i^{(t)} = \mathbf{V}_i^\top \mathbf{m}_i^{(t)}$ , and (iii)  $\mathbf{x}_i^{(t)} = \mathbf{V}_i \lambda_i^{(t)}$  hold at all  $t$ , where  $\mathbf{V}_i$  is the matrix whose columns are the vertices  $\mathcal{V}_{\Omega_i}$  (see also Figure 1). The above discussion shows that in a precise sense, Vertex OMWU and OMWU are the same algorithm, just on different equivalent representations of the game. Hence, the regret cumulated by each player  $i$  in  $\Gamma$  matches the regret cumulated by the same player in  $\tilde{\Gamma}$ , showing that Properties 2.2 and 2.3 hold for Vertex OMWU. Furthermore, whenever  $\lambda_i^{(t)}$  converges in iterates, then clearly so does  $\mathbf{x}_i^{(t)} = \mathbf{V}_i \lambda_i^{(t)}$ , showing that Property 2.4 applies to Vertex OMWU as well.

The main drawback of Vertex OMWU is that it is not clear how to avoid a per-iteration complexity linear in the number of vertices of  $\Omega$ , which is typically exponential in  $d$  (this is the case in extensive-form games). While different learning algorithms that guarantee polynomial per-iteration complexity in  $d$  exist, none of them is known to guarantee near-optimal per-player regret (Property 2.2) or last-iterate convergence (Property 2.4) enjoyed by Vertex OMWU, much less all three Properties 2.2 to 2.4 at the same time. In the rest of the paper we fill this gap, by showing that in several cases of interest, Vertex OMWU can be implemented with polynomial-time (in  $d$ ) iterations using a kernel trick.

#### 4. Kernelized Multiplicative Weights Update

In this section, we introduce *Kernelized OMWU (KOMWU)*. Kernelized OMWU gives a way of efficiently simulating the

Vertex OMWU algorithm described in Section 3 on polyhedral decision sets whose vertices have 0/1 integer coordinates, as long as a specific *polyhedral kernel* function can be evaluated efficiently. We will assume that we are given a polytope  $\Omega \subseteq \mathbb{R}^d$  with (possibly exponentially many) 0/1 integral vertices  $\mathcal{V}_\Omega := \{\mathbf{v}_1, \dots, \mathbf{v}_{|\mathcal{V}_\Omega|}\} \subseteq \{0, 1\}^d$ . Furthermore, given a vertex  $\mathbf{v} \in \mathcal{V}_\Omega$ , we will write  $k \in \mathbf{v}$  as a shorthand for  $\mathbf{v}[k] = 1$ .

We define the *0/1-polyhedral feature map*  $\phi_\Omega : \mathbb{R}^d \rightarrow \mathbb{R}^{\mathcal{V}_\Omega}$  associated with  $\Omega$  as the function such that

$$\phi_\Omega(\mathbf{x})[\mathbf{v}] := \prod_{k \in \mathbf{v}} \mathbf{x}[k] \quad \forall \mathbf{x} \in \mathbb{R}^d, \mathbf{v} \in \mathcal{V}_\Omega. \quad (3)$$

Correspondingly, the *0/1-polyhedral kernel*  $K_\Omega$  associated with  $\Omega$  is defined as the function  $K_\Omega : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ ,

$$K_\Omega(\mathbf{x}, \mathbf{y}) := \langle \phi_\Omega(\mathbf{x}), \phi_\Omega(\mathbf{y}) \rangle = \sum_{\mathbf{v} \in \mathcal{V}_\Omega} \prod_{k \in \mathbf{v}} \mathbf{x}[k] \mathbf{y}[k]. \quad (4)$$

We show that Vertex OMWU can be simulated using  $d + 1$  evaluation of the kernel  $K_\Omega$  at every iteration. The key observation is summarized in the next theorem, which shows that the iterates  $\lambda^{(t)}$  produced by Vertex OMWU are highly structured, in the sense that they are always proportional to the feature mapping  $\phi_\Omega(\mathbf{b}^{(t)})$  for some  $\mathbf{b}^{(t)} \in \mathbb{R}^d$ .

**Theorem 4.1.** *Consider the Vertex OMWU algorithm (♣), (♠). At all times  $t \geq 0$ , the vector  $\mathbf{b}^{(t)} \in \mathbb{R}^d$  defined as*

$$\mathbf{b}^{(t)}[k] := \exp \left\{ - \sum_{\tau=1}^t \eta^{(\tau)} \mathbf{w}^{(\tau)}[k] \right\} \quad (5)$$

for all  $k = 1, \dots, d$ , is such that

$$\lambda^{(t)} = \frac{\phi_\Omega(\mathbf{b}^{(t)})}{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1})}. \quad (6)$$

*Proof.* By induction.

- At time  $t = 0$ , the vector  $\mathbf{b}^{(0)}$  is  $\mathbf{b}^{(0)} = \mathbf{1} \in \mathbb{R}^d$ . By definition of the feature map (3),  $\phi_\Omega(\mathbf{1}) = \mathbf{1} \in \mathbb{R}^{\mathcal{V}_\Omega}$ . So,  $K_\Omega(\mathbf{b}^{(0)}, \mathbf{1}) = \sum_{\mathbf{v} \in \mathcal{V}_\Omega} 1 = |\mathcal{V}_\Omega|$  and hence the right-hand side of (6) is  $\frac{1}{|\mathcal{V}_\Omega|} \mathbf{1}$ , which matches  $\lambda^{(0)}$  produced by Vertex OMWU, as we wanted to show.
- Assume the statement holds up to some time  $t - 1 \geq 0$ . We will show that it holds at time  $t$  as well. Since  $\mathbf{v}$  has integral 0/1 coordinates, we can write

$$\begin{aligned} \exp\{-\eta^{(t)} \langle \mathbf{w}^{(t)}, \mathbf{v} \rangle\} &= \exp \left\{ -\eta^{(t)} \sum_{k \in \mathbf{v}} \mathbf{w}^{(t)}[k] \right\} \\ &= \prod_{k \in \mathbf{v}} \exp\{-\eta^{(t)} \mathbf{w}^{(t)}[k]\}. \end{aligned} \quad (7)$$

From the inductive hypothesis and (3), for all  $\mathbf{v} \in \mathcal{V}_\Omega$ ,

$$\lambda^{(t-1)}[\mathbf{v}] = \frac{\phi_\Omega(\mathbf{b}^{(t-1)})[\mathbf{v}]}{K_\Omega(\mathbf{b}^{(t-1)}, \mathbf{1})} = \frac{\prod_{k \in \mathbf{v}} \mathbf{b}^{(t-1)}[k]}{K_\Omega(\mathbf{b}^{(t-1)}, \mathbf{1})}. \quad (8)$$

Plugging (7) and (8) into (♣), we have the inductive step

$$\begin{aligned}\lambda^{(t)}[\mathbf{v}] &= \frac{\prod_{k \in \mathbf{v}} \mathbf{b}^{(t-1)}[k] \exp\{-\eta^{(t)} \mathbf{w}^{(t)}[k]\}}{\sum_{\mathbf{v} \in \mathcal{V}_\Omega} \prod_{k \in \mathbf{v}} \mathbf{b}^{(t-1)}[k] \exp\{-\eta^{(t)} \mathbf{w}^{(t)}[k]\}} \\ &= \frac{\phi_\Omega(\mathbf{b}^{(t)})[\mathbf{v}]}{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1})}\end{aligned}$$

for all  $\mathbf{v} \in \mathcal{V}_\Omega$ , where in the last step we used the fact that  $\mathbf{b}^{(t)}[k] = \mathbf{b}^{(t-1)}[k] \exp\{-\eta^{(t)} \mathbf{w}^{(t)}[k]\}$  by (5).  $\square$

The structure of  $\lambda^{(t)}$  uncovered by Theorem 4.1 can be leveraged to compute the iterate  $\mathbf{x}^{(t)}$  produced by Vertex OMWU, *i.e.*, the convex combination of the vertices (♠), using  $d + 1$  evaluations of the kernel  $K_\Omega$ . We do so by extending an idea of Takimoto & Warmuth (2003, eq. 5.2).

**Theorem 4.2.** *Let  $\mathbf{b}^{(t)}$  be as in Theorem 4.1. For each  $h = 1, \dots, d$ , let  $\bar{\mathbf{e}}_h \in \mathbb{R}^d$  be defined as the indicator vector*

$$\bar{\mathbf{e}}_h[k] := \mathbb{1}_{k \neq h} := \begin{cases} 0 & \text{if } k = h \\ 1 & \text{if } k \neq h. \end{cases} \quad (9)$$

Then, at all  $t \geq 1$ , the iterate  $\mathbf{x}^{(t)} \in \Omega$  produced by Vertex OMWU can be written as

$$\mathbf{x}^{(t)} = \left(1 - \frac{K_\Omega(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_1)}{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1})}, \dots, 1 - \frac{K_\Omega(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_d)}{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1})}\right). \quad (10)$$

*Proof.* The proof crucially relies on the observation that for all  $h = 1, \dots, d$ , the feature map  $\phi_\Omega(\bar{\mathbf{e}}_h)$  satisfies

$$\phi_\Omega(\bar{\mathbf{e}}_h)[\mathbf{v}] = \prod_{k \in \mathbf{v}} \bar{\mathbf{e}}_h[k] = \prod_{k \in \mathbf{v}} \mathbb{1}_{k \neq h} = \mathbb{1}_{h \notin \mathbf{v}}, \quad \forall \mathbf{v} \in \mathcal{V}_\Omega.$$

Using the fact that  $\phi_\Omega(\mathbf{1}) = \mathbf{1}$ , we conclude that

$$\phi_\Omega(\mathbf{1})[\mathbf{v}] - \phi_\Omega(\bar{\mathbf{e}}_h)[\mathbf{v}] = \mathbb{1}_{h \in \mathbf{v}}, \quad \forall h = 1, \dots, d. \quad (11)$$

Therefore, for all  $k = 1, \dots, d$ , we obtain

$$\begin{aligned}\mathbf{x}^{(t)}[k] &\stackrel{(\clubsuit)}{=} \sum_{\mathbf{v} \in \mathcal{V}_\Omega} \lambda^{(t)}[\mathbf{v}] \cdot \mathbf{v}[k] = \sum_{\mathbf{v} \in \mathcal{V}_\Omega} \lambda^{(t)}[\mathbf{v}] \cdot \mathbb{1}_{k \in \mathbf{v}} \\ &= \sum_{\mathbf{v} \in \mathcal{V}_\Omega} \lambda^{(t)}[\mathbf{v}] \cdot (\phi_\Omega(\mathbf{1})[\mathbf{v}] - \phi_\Omega(\bar{\mathbf{e}}_k)[\mathbf{v}]) \\ &= \frac{\langle \phi_\Omega(\mathbf{b}^{(t)}), \phi_\Omega(\mathbf{1}) \rangle - \langle \phi_\Omega(\mathbf{b}^{(t)}), \phi_\Omega(\bar{\mathbf{e}}_k) \rangle}{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1})} \\ &= \frac{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1}) - K_\Omega(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_k)}{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1})} = 1 - \frac{K_\Omega(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_k)}{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1})},\end{aligned}$$

where the second equality follows from the integrality of  $\mathbf{v} \in \mathcal{V}_\Omega$ , the third from (11), the fourth from Theorem 4.1, and the fifth from the definition of  $K_\Omega$  (4).  $\square$

---

**Algorithm 1: Kernelized OMWU (KOMWU)**


---

```

1  $\ell^{(0)}, \mathbf{m}^{(0)}, \mathbf{s}^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^d$  [ $\triangleright$  Initialization]
2 for  $t = 1, 2, \dots$  do
3   receive prediction  $\mathbf{m}^{(t)} \in \mathbb{R}^d$  of next loss
4   [ $\triangleright$  set  $\mathbf{m}^{(t)} = \mathbf{0}$  for non-predictive variant]
5   [ $\triangleright$  Compute  $\mathbf{b}^{(t)}$  according to Theorem 4.1]
6    $\mathbf{w}^{(t)} \leftarrow \ell^{(t-1)} - \mathbf{m}^{(t-1)} + \mathbf{m}^{(t)}$ 
7    $\mathbf{s}^{(t)} \leftarrow \mathbf{s}^{(t-1)} + \eta^{(t)} \mathbf{w}^{(t)}$  [ $\triangleright \mathbf{s}^{(t)} = \sum \eta^{(\tau)} \mathbf{w}^{(\tau)}$ ]
8   for  $k = 1, \dots, d$  do
9     [ $\mathbf{b}^{(t)}[k] \leftarrow \exp\{-\mathbf{s}^{(t)}[k]\}$ ] [ $\triangleright$  see (5)]
10  [ $\triangleright$  Produce iterate  $\mathbf{x}^{(t)}$  according to Theorem 4.2]
11   $\mathbf{x}^{(t)} \leftarrow \mathbf{0} \in \mathbb{R}^d$ 
12   $\alpha \leftarrow K_\Omega(\mathbf{b}^{(t)}, \mathbf{1})$  [ $\triangleright K_\Omega$  is defined in (4)]
13  for  $k = 1, \dots, d$  do
14    [ $\mathbf{x}^{(t)}[k] \leftarrow 1 - K_\Omega(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_k) / \alpha$ ] [ $\triangleright$  see (10)]
15  output  $\mathbf{x}^{(t)} \in \Omega$  and receive loss vector
16   $\ell^{(t)} \in \mathbb{R}^d$ 

```

---

Combined, Theorems 4.1 and 4.2 suggest that by keeping track of the vectors  $\mathbf{b}^{(t)}$  instead of  $\lambda^{(t)}$ , updating them using Theorem 4.1 and reconstructing the iterates  $\mathbf{x}^{(t)}$  using Theorem 4.2, Vertex OMWU can be simulated efficiently. We call the resulting algorithm, given in Algorithm 1, *Kernelized OMWU (KOMWU)*. Similarly, we call *Kernelized MWU* the non-optimistic version of KOMWU obtained as the special case in which  $\mathbf{m}^{(t)} = \mathbf{0}$  at all  $t$ . In light of the preceding discussion, we have the following.

**Theorem 4.3.** *Kernelized OMWU produces the same iterates  $\mathbf{x}^{(t)}$  as Vertex OMWU when it receives the same sequence of predictions  $\mathbf{m}^{(t)}$  and losses  $\ell^{(t)} \in \mathbb{R}^d$ . Furthermore, each iteration of KOMWU runs in time proportional to the time required to compute the  $d + 1$  kernel evaluations  $\{K_\Omega(\mathbf{b}^{(t)}, \mathbf{1}), K_\Omega(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_1), \dots, K_\Omega(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_d)\}$ .*

## 5. KOMWU in Extensive-Form Games

In this section, we show how the general theory we developed in Section 5 applies to extensive-form game, *i.e.*, tree-form games that incorporate sequential and simultaneous moves, and imperfect information. The central result of this section, Theorem 5.4, shows that OMWU on the normal-form representation of any EFG can be simulated in linear time in the game tree size via KOMWU, contradicting the popular wisdom that working with the normal form of an extensive-form game is intractable.

### 5.1. Preliminaries on Extensive-Form Games

We now briefly recall standard concepts and notation about extensive-form games which we use in the rest of the section. More details and an example are available in Appendix C.

In an  $m$ -player perfect-recall extensive-form game, each player  $i \in \llbracket m \rrbracket$  faces a tree-form sequential decision problem (TFSDP). In a TFSDP, the player interacts with the environment in two ways: at *decision points*, the agent must act by picking an action from a set of legal actions; at *observation points*, the agent observes a signal drawn from a set of possible signals. We denote the set of decision points of player  $i$  as  $\mathcal{J}_i$ . The set of actions available at decision point  $j \in \mathcal{J}_i$  is denoted  $A_j$ . A pair  $(j, a)$  where  $j \in \mathcal{J}_i$  and  $a \in A_j$  is called a *non-empty sequence*. The set of all non-empty sequences of player  $i$  is denoted as  $\Sigma_i^* := \{(j, a) : j \in \mathcal{J}_i, a \in A_j\}$ . For notational convenience, we will often denote an element  $(j, a)$  in  $\Sigma_i^*$  as  $ja$  without using parentheses. Given a decision point  $j \in \mathcal{J}_i$ , we denote by  $p_j$  its *parent sequence*, defined as the last sequence (that is, decision point-action pair) encountered on the path from the root of the decision process to  $j$ . If the agent does not act before  $j$  (that is,  $j$  is the root of the process or only observation points are encountered on the path from the root to  $j$ ), we let  $p_j$  be set to the special element  $\emptyset$ , called the *empty sequence*. We let  $\Sigma_i := \Sigma_i^* \cup \{\emptyset\}$ . Given a  $\sigma \in \Sigma_i$ , we let  $\mathcal{C}_\sigma := \{j \in \mathcal{J}_i : p_j = \sigma\}$ .

An  $m$ -player extensive-form game is a polyhedral convex game (Section 3)  $\Gamma = (m, \{Q_i\}, \{U_i\})$ , where the convex polytope of mixed strategies  $Q_i$  of each player  $i \in \llbracket m \rrbracket$  is called a *sequence-form strategy space* (Romanovskii, 1962; von Stengel, 1996; Koller et al., 1996), and is defined as

$$Q_i := \left\{ \mathbf{x} \in \mathbb{R}^{\Sigma_i} : \begin{array}{l} \textcircled{1} \mathbf{x}[\emptyset] = 1, \\ \textcircled{2} \mathbf{x}[p_j] = \sum_{a \in A_j} \mathbf{x}[ja] \quad \forall j \in \mathcal{J}_i \end{array} \right\}.$$

It is known that the set of vertices of  $Q_i$  are the *deterministic* sequence-form strategies  $\Pi_i := Q_i \cap \{0, 1\}^{\Sigma_i}$ . We mention the following result (see Appendix E).

**Proposition 5.1.** *The number of vertices of  $Q_i$  is upper bounded by  $A^{\|Q_i\|_1}$ , where  $A := \max_{j \in \mathcal{J}_i} |A_j|$  is the largest number of possible actions, and  $\|Q_i\|_1 := \max_{\mathbf{q} \in Q_i} \|\mathbf{q}\|_1$ .*

We will often need to describe strategies for *subtrees* of the TFSDM faced by each player  $i$ . We use the notation  $j' \succeq j$  to denote the fact that  $j' \in \mathcal{J}_i$  is a descendant of  $j \in \mathcal{J}_i$ , and  $j' \succ j$  to denote a strict descendant (i.e.,  $j' \succeq j \wedge j' \neq j$ ). For any  $j \in \mathcal{J}_i$  we let  $\Sigma_{i,j}^* := \{j'a' : j' \succeq j, a' \in A_{j'}\}$  denote the set of non-empty sequences in the subtree rooted at  $j$ . The set of sequence-form strategies for that subtree  $j$  is defined as the convex polytope

$$Q_{i,j} := \left\{ \mathbf{x} \in \mathbb{R}^{\Sigma_{i,j}^*} : \begin{array}{l} \textcircled{1} \sum_{a \in A_j} \mathbf{x}[ja] = 1, \\ \textcircled{2} \mathbf{x}[p_{j'}] = \sum_{a \in A_{j'}} \mathbf{x}[j'a] \quad \forall j' \succ j \end{array} \right\}.$$

Correspondingly, we let  $\Pi_{i,j} := Q_{i,j} \cap \{0, 1\}^{\Sigma_{i,j}^*}$  denote the set of vertices of  $Q_{i,j}$ , each of which is a deterministic sequence-form strategy for the subtree rooted at  $j$ .

## 5.2. Linear-time Implementation of KOMWU

For any player  $i$ , the 0/1-polyhedral kernel  $K_{Q_i}$  associated with the player's sequence-form strategy space  $Q_i$  can be evaluated in linear time in the number of sequences  $|\Sigma_i|$  of that player. To do so, we introduce a *partial kernel function*  $K_j : \mathbb{R}^{\Sigma_i} \times \mathbb{R}^{\Sigma_i} \rightarrow \mathbb{R}$  for every decision point  $j \in \mathcal{J}_i$ ,

$$K_j(\mathbf{x}, \mathbf{y}) := \sum_{\pi \in \Pi_{i,j}} \prod_{\sigma \in \pi} \mathbf{x}[\sigma] \mathbf{y}[\sigma]. \quad (12)$$

**Theorem 5.2.** *For any vectors  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{\Sigma_i}$ , the two following recursive relationships hold:*

$$K_{Q_i}(\mathbf{x}, \mathbf{y}) = \mathbf{x}[\emptyset] \mathbf{y}[\emptyset] \prod_{j \in \mathcal{C}_\emptyset} K_j(\mathbf{x}, \mathbf{y}), \quad (13)$$

and, for all decision points  $j \in \mathcal{J}_i$ ,

$$K_j(\mathbf{x}, \mathbf{y}) = \sum_{a \in A_j} \left( \mathbf{x}[ja] \mathbf{y}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{y}) \right). \quad (14)$$

In particular, Equations (13) and (14) give a recursive algorithm to evaluate the polyhedral kernel  $K_{Q_i}$  associated with the sequence-form strategy space of any player  $i$  in an EFG in linear time in the number of sequences  $|\Sigma_i|$ .

Theorem 5.2 shows that the kernel  $K_{Q_i}$  can be evaluated in linear time (in  $|\Sigma_i|$ ) at any  $(\mathbf{x}, \mathbf{y})$ . So, the KOMWU algorithm (Algorithm 1) can be trivially implemented for  $\Omega = Q_i$  in quadratic  $\mathcal{O}(|\Sigma_i|^2)$  time per iteration by directly evaluating the  $|\Sigma_i| + 1$  kernel evaluations  $\{K_{Q_i}(\mathbf{b}^{(t)}, \mathbf{1})\} \cup \{K_{Q_i}(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_\sigma) : \sigma \in \Sigma_i\}$  needed at each iteration, where  $\bar{\mathbf{e}}_\sigma \in \mathbb{R}^{\Sigma_i}$ , defined in (9) for the general case, is the vector whose components are  $\bar{\mathbf{e}}_\sigma[\sigma'] := \mathbb{1}_{\sigma \neq \sigma'}$  for all  $\sigma, \sigma' \in \Sigma_i$ . We refine that result by showing that an implementation of KOMWU with *linear-time* (i.e.,  $\mathcal{O}(|\Sigma_i|)$ ) per-iteration complexity exists, by exploiting the structure of the particular set of kernel evaluations needed at every iteration. In particular, we rely on the following observation.

**Proposition 5.3.** *For any player  $i \in \llbracket m \rrbracket$ , vector  $\mathbf{x} \in \mathbb{R}_{>0}^{\Sigma_i}$ , and sequence  $ja \in \Sigma_i^*$ ,*

$$\frac{1 - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{ja}) / K_{Q_i}(\mathbf{x}, \mathbf{1})}{1 - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{p_j}) / K_{Q_i}(\mathbf{x}, \mathbf{1})} = \frac{\mathbf{x}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{1})}{K_j(\mathbf{x}, \mathbf{1})}.$$

In order to compute  $\{K_{Q_i}(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_\sigma) : \sigma \in \Sigma_i\}$  in cumulative  $\mathcal{O}(|\Sigma_i|)$  time, we then do the following.

1. We compute the values  $K_j(\mathbf{b}^{(t)}, \mathbf{1})$  for all  $j \in \mathcal{J}_i$  in cumulative  $\mathcal{O}(|\Sigma_i|)$  time by using (14).
2. We compute the ratio  $K_{Q_i}(\mathbf{b}^{(t)}, \bar{\mathbf{e}}_\sigma) / K_{Q_i}(\mathbf{b}^{(t)}, \mathbf{1})$  by evaluating the two kernel separately using Theorem 5.2, spending  $\mathcal{O}(|\Sigma_i|)$  time.

3. We repeatedly use Proposition 5.3 in a top-down fashion along the tree-form decision problem of player  $i$  to compute the ratio  $K_{Q_i}(\mathbf{b}^{(t)}, \bar{e}_{ja})/K_{Q_i}(\mathbf{b}^{(t)}, \mathbf{1})$  for each sequence  $ja \in \Sigma_i^*$  given the value of the parent ratio  $K_{Q_i}(\mathbf{b}^{(t)}, \bar{e}_{pj})/K_{Q_i}(\mathbf{b}^{(t)}, \mathbf{1})$  and the partial kernel evaluations  $\{K_j(\mathbf{b}^{(t)}, \mathbf{1}) : j \in \mathcal{J}_i\}$  from Step 1. For each  $ja \in \Sigma_i^*$ , Proposition 5.3 gives a formula whose runtime is linear in the number of children decision points  $|\mathcal{C}_{ja}|$  at that sequence. Therefore, the cumulative runtime required to compute all ratios  $K_{Q_i}(\mathbf{b}^{(t)}, \bar{e}_{ja})/K_{Q_i}(\mathbf{b}^{(t)}, \mathbf{1})$  is  $\mathcal{O}(|\Sigma_i|)$ .
4. By multiplying the ratios computed in Step 3 by the value of  $K_{Q_i}(\mathbf{b}^{(t)}, \mathbf{1})$  computed in Step 2, we can easily recover each  $K_{Q_i}(\mathbf{b}^{(t)}, \bar{e}_\sigma)$  for every  $\sigma \in \Sigma_i^*$ .

Hence, we have just proved the following.

**Theorem 5.4.** *For each player  $i$  in a perfect-recall extensive-form game, the Kernelized OMWU algorithm can be implemented exactly, with a per-iteration complexity linear in the number of sequences  $|\Sigma_i|$  of that player.*

### 5.3. KOMWU Regret Bounds and Convergence

If the players in an EFG run KOMWU, then we can combine Theorem 4.3 with standard OMWU regret bounds, Proposition 5.1 and Properties 2.2 to 2.4 to get the following:

**Theorem 5.5.** *In an EFG, after  $T$  rounds of learning under the COLS, KOMWU satisfies*

1. A player  $i$  using KOMWU with  $\eta^{(t)} := \eta = \sqrt{8 \log(A) \|Q_i\|_1 / \sqrt{T}}$  is guaranteed to incur regret at most  $R_i^T = \mathcal{O}(\sqrt{\|Q_i\|_1 \log(A) T})$ .
2. There exist  $C, C' > 0$  such that if all  $m$  players learn using KOMWU with constant learning rate  $\eta^{(t)} := \eta \leq 1/(Cm \log^4 T)$ , then each player is guaranteed to incur regret at most  $\frac{\log(A_i) \|Q_i\|_1}{\eta} + C' \log T$ .
3. If all  $m$  player learn using KOMWU with  $\eta^{(t)} := \eta \leq 1/\sqrt{8(m-1)}$ , then the sum of regrets is at most  $\sum_{i=1}^m R_i^T = \mathcal{O}(\max_{i=1}^m \{\|Q_i\|_1 \log A_i\}^{\frac{m}{\eta}})$ .
4. For two-player zero-sum EFGs, if both players learn using KOMWU, then there exists a schedule of learning rates  $\eta^{(t)}$  such that the iterates converge to a Nash equilibrium. Furthermore, if the NFG representation of the EFG has a unique Nash equilibrium and both players use learning rates  $\eta^{(t)} = \eta \leq 1/8$ , then the iterates converge to a Nash equilibrium at a linear rate  $C(1+C')^{-t}$ , where  $C, C'$  are constants that depend on the game.

Prior to our result, the strongest regret bound for methods that take linear time per iteration was based on instantiating e.g. follow the regularized leader (FTRL) or its optimistic variant with the dilatable global entropy regularizer of Farina et al. (2021a). For FTRL this yields a regret bound of the form  $\mathcal{O}(\sqrt{\log(A) \|Q\|_1^2 T})$ . For optimistic FTRL this yields

a regret bound of the form  $\mathcal{O}(\log(A) \|Q\|_1^2 \sqrt{mT}^{1/4})$ , when every player in an  $m$ -player game uses that algorithm and appropriate learning rates.

Our algorithm improves the state-of-the-art rate in two ways. First, we improve the dependence on game constants by almost a square root factor, because our dependence on  $\|Q\|_1$  is smaller by a square root, compared to prior results. Secondly, in the multi-player general-sum setting, every other method achieves regret that is on the order of  $T^{1/4}$ , whereas our method achieves regret on the order of  $\log^4(T)$ . In the context of two-player zero-sum EFGs, the bound on the sum of regrets in Theorem 5.5 guarantees convergence to a Nash equilibrium at a rate of  $\mathcal{O}(\max_i \|Q_i\|_1 \log A_i / T)$ . This similarly improves the prior state of the art.

Lee et al. (2021) showed the first last-iterate results for EFGs using algorithms that require linear time per iteration. In particular, they show that the dilated entropy DGF combined with optimistic online mirror descent leads to last-iterate convergence at a linear rate. However, their result requires learning rates  $\eta \leq 1/(8|\Sigma_i|)$ . This learning rate is impractically small in practice. In contrast, our last-iterate linear-rate result for KOMWU allows learning rates of size  $1/8$ . That said, our result is not directly comparable to theirs. The existence of a unique Nash equilibrium in the EFG representation is a necessary condition for uniqueness in the NFG representation. However, it is possible that the NFG has additional equilibria even when the EFG does not. Wei et al. (2021) conjecture that linear-rate convergence holds even without the assumption of a unique Nash equilibrium. If this conjecture turns out to be true for NFGs, then Theorem 4.3 would immediately imply that KOMWU also has last-iterate linear-rate convergence without the uniqueness assumption.

### 5.4. Experimental Evaluation

We numerically investigate agents learning under the COLS in Kuhn and Leduc poker (Kuhn, 1950; Southey et al., 2005). We compare the maximum per-player regret cumulated by KOMWU for four different choices of constant learning rate, against that cumulated by two standard algorithms from the extensive-form game solving literature (CFR and CFR(RM+)). More details about the games and the algorithms are given in Appendix D. Results are shown in Figure 2. We observe that the per-player regret cumulated by KOMWU plateaus and remains constants, unlike the CFR variants. This behavior is consistent with the near-optimal per-player regret guarantees of KOMWU (Theorem 5.5).

## 6. Conclusions

We introduce the Kernelized OMWU algorithm for simulating OMWU on the vertices of a 0/1-polyhedral set. KOMWU can be implemented via black-box access to ker-

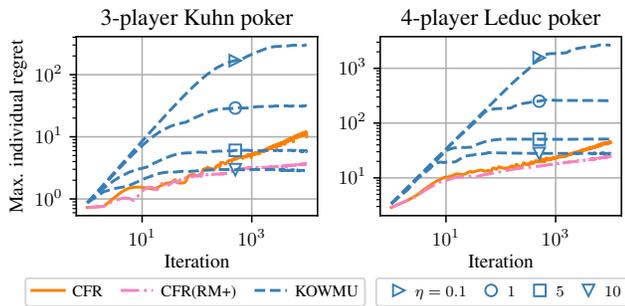


Figure 2. Maximum per-player regret cumulated by KOWMU compared to two variants of the CFR algorithm.

nel evaluations, and these evaluations can be performed in linear time for EFGs. This leads to new state-of-the-art regret bounds and other properties for no-regret learning on EFGs that were previously only obtained for NFGs. In the appendix, we show that KOMWU can be implemented efficiently for several other domains:  $n$ -sets, which are 0/1-polyhedral sets of the form  $\pi \in \{0, 1\}^d : \|\pi\|_1 = n$ , the unit hypercube, flows in directed acyclic graphs, permutations, and Cartesian products of sets with efficient kernel evaluations. For  $n$ -sets we obtain an improved cost-per-iteration compared to existing methods for simulating OMWU.

## Acknowledgments

This material is based on work supported by the National Science Foundation under grants IIS-1718457, IIS-1901403, IIS-1943607, and CCF-1733556, and the ARO under award W911NF2010081.

## References

- Audibert, J.-Y., Bubeck, S., and Lugosi, G. Regret in online combinatorial optimization. *Mathematics of Operations Research*, 39(1):31–45, 2014.
- Bowling, M., Burch, N., Johanson, M., and Tammelin, O. Heads-up limit hold’em poker is solved. *Science*, 347(6218), January 2015.
- Brown, N. and Sandholm, T. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, pp. eaao1733, Dec. 2017.
- Brown, N. and Sandholm, T. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, 2019.
- Cesa-Bianchi, N. and Lugosi, G. Combinatorial bandits. *Journal of Computer and System Sciences*, 78(5):1404–1422, 2012.
- Chen, X. and Peng, B. Hedging in games: Faster convergence of external and swap regrets. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- Daskalakis, C. and Panageas, I. Last-iterate convergence: Zero-sum games and constrained min-max optimization. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- Daskalakis, C., Fishelson, M., and Golowich, N. Near-optimal no-regret learning in general games. *CoRR*, abs/2108.06924, 2021.
- Farina, G., Celli, A., Gatti, N., and Sandholm, T. Ex ante coordination and collusion in zero-sum multi-player extensive-form games. In *Advances in Neural Information Processing Systems*, pp. 9638–9648, 2018.
- Farina, G., Kroer, C., Brown, N., and Sandholm, T. Stable-predictive optimistic counterfactual regret minimization. In *International Conference on Machine Learning (ICML)*, 2019a.
- Farina, G., Kroer, C., and Sandholm, T. Optimistic regret minimization for extensive-form games via dilated distance-generating functions. In *Advances in Neural Information Processing Systems, NeurIPS 2019*, pp. 5222–5232, 2019b.
- Farina, G., Kroer, C., and Sandholm, T. Better regularization for sequential decision spaces: Fast convergence rates for Nash, correlated, and team equilibria. In *ACM Conference on Economics and Computation*, 2021a.
- Farina, G., Kroer, C., and Sandholm, T. Faster game solving via predictive blackwell approachability: Connecting regret matching and mirror descent. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021b.
- Gordon, G. J., Greenwald, A., and Marks, C. No-regret learning in convex games. In *Proceedings of the 25th international conference on Machine learning*, pp. 360–367. ACM, 2008.
- Helmbold, D. and Warmuth, M. Learning permutations with exponential weights. *Journal of Machine Learning Research*, 10(7), 2009.
- Hoda, S., Gilpin, A., Peña, J., and Sandholm, T. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2), 2010.
- Hsieh, Y.-G., Antonakopoulos, K., and Mertikopoulos, P. Adaptive learning in continuous games: Optimal regret bounds and convergence to nash equilibrium. *arXiv preprint arXiv:2104.12761*, 2021.

- Jerrum, M., Sinclair, A., and Vigoda, E. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM (JACM)*, 51(4):671–697, 2004.
- Kalai, A. and Vempala, S. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71:291–307, 2005.
- Koller, D., Megiddo, N., and von Stengel, B. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2), 1996.
- Koo, T., Globerson, A., Carreras Pérez, X., and Collins, M. Structured prediction models via the matrix-tree theorem. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 141–150, 2007.
- Koolen, W. M., Warmuth, M. K., and Kivinen, J. Hedging structured concepts. In *COLT 2010: Proceedings of the 23rd Annual Conference on Learning Theory*, pp. 93–105, 2010.
- Kroer, C., Waugh, K., Kılınc-Karzan, F., and Sandholm, T. Faster first-order methods for extensive-form game solving. In *Proceedings of the ACM Conference on Economics and Computation (EC)*, 2015.
- Kroer, C., Farina, G., and Sandholm, T. Solving large sequential games with the excessive gap technique. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2018.
- Kroer, C., Waugh, K., Kılınc-Karzan, F., and Sandholm, T. Faster algorithms for extensive-form game solving via improved smoothing functions. *Mathematical Programming*, 2020.
- Kuhn, H. W. A simplified two-person poker. In Kuhn, H. W. and Tucker, A. W. (eds.), *Contributions to the Theory of Games*, volume 1 of *Annals of Mathematics Studies*, 24, pp. 97–103. Princeton University Press, Princeton, New Jersey, 1950.
- Lee, C.-W., Kroer, C., and Luo, H. Last-iterate convergence in extensive-form games. *Advances in Neural Information Processing Systems*, 34, 2021.
- Lei, Q., Nagarajan, S. G., Panageas, I., et al. Last iterate convergence in no-regret learning: constrained min-max optimization for convex-concave landscapes. In *International Conference on Artificial Intelligence and Statistics*, pp. 1441–1449. PMLR, 2021.
- Moravčík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, May 2017.
- Rakhlin, A. and Sridharan, K. Online learning with predictable sequences. In *Conference on Learning Theory*, pp. 993–1019, 2013a.
- Rakhlin, A. and Sridharan, K. Optimization, learning, and games with predictable sequences. In *Advances in Neural Information Processing Systems*, pp. 3066–3074, 2013b.
- Romanovskii, I. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3, 1962.
- Southey, F., Bowling, M., Larson, B., Piccione, C., Burch, N., Billings, D., and Rayner, C. Bayes’ bluff: Opponent modelling in poker. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2005.
- Syrkkanis, V., Agarwal, A., Luo, H., and Schapire, R. E. Fast convergence of regularized learning in games. In *Advances in Neural Information Processing Systems*, pp. 2989–2997, 2015.
- Takimoto, E. and Warmuth, M. K. Path kernels and multiplicative updates. *The Journal of Machine Learning Research*, 4:773–818, 2003.
- Tammelin, O., Burch, N., Johanson, M., and Bowling, M. Solving heads-up limit Texas hold’em. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.
- von Stengel, B. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.
- Warmuth, M. K. and Kuzmin, D. Randomized online pca algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9 (Oct):2287–2320, 2008.
- Wei, C.-Y., Lee, C.-W., Zhang, M., and Luo, H. Linear last-iterate convergence in constrained saddle-point optimization. In *International Conference on Learning Representations*, 2021.
- Zinkevich, M., Bowling, M., Johanson, M., and Piccione, C. Regret minimization in games with incomplete information. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.

## A. Additional Related Work

### A.1. More Results for Optimistic Algorithms in Games

For individual regret in multi-player general-sum NFGs, [Syrkkanis et al. \(2015\)](#) first show  $\mathcal{O}(T^{1/4})$  regret for general optimistic OMD and FTRL algorithms. The result is improved to  $\mathcal{O}(T^{1/6})$  by [Chen & Peng, 2020](#), but only for OMWU in two-player NFGs. [Daskalakis et al. \(2021\)](#) show that OMWU enjoys  $\mathcal{O}(\log^4 T)$  regret in multi-player general-sum NFGs.

As for last-iterate convergence in two-player zero-sum games, [Daskalakis & Panageas \(2019\)](#) show an asymptotic result for OMWU under the unique Nash equilibrium assumption. [Wei et al. \(2021\)](#) further show a linear convergence rate while allowing larger learning rates under the same assumption. [Hsieh et al. \(2021\)](#) show another asymptomatic convergence result without the assumption. It is also worth noting that OGDA, another popular optimistic algorithm, has been shown its last-iterate convergence in general polyhedron games ([Wei et al., 2021](#)).

### A.2. Approaches in Online Combinatorial Optimization

Besides performing MWU/OMWU over vertices, we review two additional approaches in online combinatorial optimization:

**OMD over the Convex Hull** This approach is running Online Mirror Descent (OMD) over the convex hull ([Koolen et al., 2010](#); [Audibert et al., 2014](#)). It is well known that OMD with the negative entropy regularizer results in a (dimension-wise) multiplicative weight update. For the case that the set of vertices is a standard basis, this algorithm coincides with the MWU over the probability simplex. However, for general cases, it requires to project back to the convex hull and the procedure may not be efficient. [Helmbold & Warmuth \(2009\)](#) first used this approach for permutations, and [Koolen et al. \(2010\)](#) generally studied it for arbitrary 0/1 polyhedral sets and show its efficiency for more cases.

**FTPL** Another approach is called Follow the Perturbed Leader ([Kalai & Vempala, 2005](#)). This approach adds a random perturbation to the cumulative loss vector, and greedily selects the vertex with minimal perturbed loss. The latter procedure corresponds to linear optimization over the set of vertices, which can be solved efficiently for most cases of interest. We are not aware of any previous work using this approach for EFGs though.

## B. Pseudocode

Below we show pseudocode for OMWU and Vertex OMWU (Section 3).

---

### Algorithm 2: OMWU

---

**Data:** Finite set of choices  $\mathcal{A}$ , learning rates  $\eta^{(t)} > 0$

```

1  $\ell^{(0)}, \mathbf{m}^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^{\mathcal{A}}; \boldsymbol{\lambda}^{(0)} \leftarrow \frac{1}{|\mathcal{A}|} \mathbf{1} \in \Delta(\mathcal{A})$ 
2 for  $t = 1, 2, \dots$  do
3   receive prediction  $\mathbf{m}^{(t)} \in \mathbb{R}^{\mathcal{A}}$  of next loss
4   [ $\triangleright$  set  $\mathbf{m}^{(t)} = \mathbf{0}$  for non-predictive variant]
5    $\mathbf{w}^{(t)} \leftarrow \ell^{(t-1)} - \mathbf{m}^{(t-1)} + \mathbf{m}^{(t)}$ 
6   for  $a \in \mathcal{A}$  do
7      $\lambda^{(t)}[a] \leftarrow \frac{\lambda^{(t-1)}[a] \cdot e^{-\eta^{(t)} \mathbf{w}^{(t)}[a]}}{\sum_{a' \in \mathcal{A}} \lambda^{(t-1)}[a'] \cdot e^{-\eta^{(t)} \mathbf{w}^{(t)}[a'']}}$ 
8   output  $\boldsymbol{\lambda}^{(t)} \in \Delta(\mathcal{A})$ 
9   receive loss vector  $\ell^{(t)} \in \mathbb{R}^{\mathcal{A}}$ 
```

---



---

### Algorithm 3: Vertex OMWU

---

**Data:** Polytope  $\Omega \subseteq \mathbb{R}^d$  with vertices  $\{\mathbf{v}_1, \dots, \mathbf{v}_k\} =: \mathcal{V}_\Omega$ , learning rates  $\eta^{(t)} > 0$

```

1  $\ell^{(0)}, \mathbf{m}^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^d; \boldsymbol{\lambda}^{(0)} \leftarrow \frac{1}{|\mathcal{V}_\Omega|} \mathbf{1} \in \Delta(\mathcal{V}_\Omega)$ 
2 for  $t = 1, 2, \dots$  do
3   receive prediction  $\mathbf{m}^{(t)} \in \mathbb{R}^d$  of next loss
4   [ $\triangleright$  set  $\mathbf{m}^{(t)} = \mathbf{0}$  for non-predictive variant]
5    $\mathbf{w}^{(t)} \leftarrow \ell^{(t-1)} - \mathbf{m}^{(t-1)} + \mathbf{m}^{(t)}$ 
6   [ $\triangleright$  Run the OMWU update on  $\boldsymbol{\lambda}$  using  $\mathcal{A} = \mathcal{V}_\Omega$ ]
7   for  $\mathbf{v} \in \mathcal{V}_\Omega$  do
8      $\lambda^{(t)}[\mathbf{v}] \leftarrow \frac{\lambda^{(t-1)}[\mathbf{v}] \cdot e^{-\eta^{(t)} \langle \mathbf{w}^{(t)}, \mathbf{v} \rangle}}{\sum_{\mathbf{v}' \in \mathcal{V}_\Omega} \lambda^{(t-1)}[\mathbf{v}'] \cdot e^{-\eta^{(t)} \langle \mathbf{w}^{(t)}, \mathbf{v}' \rangle}}$ 
9   [ $\triangleright$  Compute new convex combination of vertices]
10   $\mathbf{x}^{(t)} \leftarrow \sum_{\mathbf{v} \in \mathcal{V}_\Omega} \lambda^{(t)}[\mathbf{v}] \cdot \mathbf{v}$ 
11  output  $\mathbf{x}^{(t)} \in \Omega$ 
12  receive loss vector  $\ell^{(t)} \in \mathbb{R}^d$ 
```

---

## C. Extensive-Form Games

In a *tree-form sequential decision process (TFSDP)* problem the agent interacts with the environment in two ways: at *decision points*, the agent must act by picking an action from a set of legal actions; at *observation points*, the agent observes a signal drawn from a set of possible signals. Different decision points can have different sets of legal actions, and different observation points can have different sets of possible signals. Decision and observation points are structured as a *tree*: under the standard assumption that the agent is not forgetful, so, it is not possible for the agent to cycle back to a previously encountered decision or observation point by following the structure of the decision problem.

As an example, consider the simplified game of *Kuhn poker* (Kuhn, 1950), depicted in Figure 3. Kuhn poker is a standard benchmark in the EFG-solving community. In Kuhn poker, each player puts an ante worth 1 into the pot. Each player is then privately dealt one card from a deck that contains 3 unique cards (Jack, Queen, King). Then, a single round of betting then occurs, with the following dynamics. First, Player 1 decides to either check or bet 1. Then,

- If Player 1 checks Player 2 can check or raise 1.
  - If Player 2 checks a showdown occurs; if Player 2 raises Player 1 can fold or call.
    - \* If Player 1 folds Player 2 takes the pot; if Player 1 calls a showdown occurs.
- If Player 1 raises Player 2 can fold or call.
  - If Player 2 folds Player 1 takes the pot; if Player 2 calls a showdown occurs.

When a showdown occurs, the player with the higher card wins the pot and the game immediately ends.

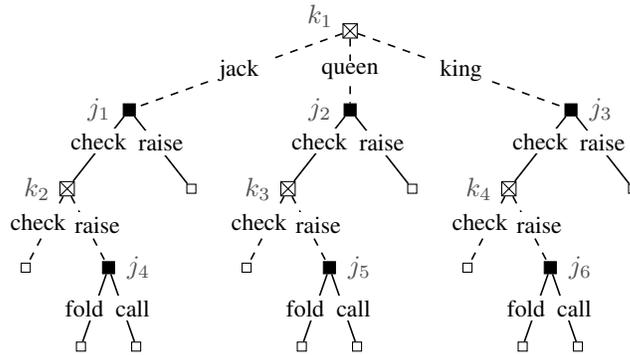


Figure 3. Tree-form sequential decision making process of the first acting player in the game of Kuhn poker.

As soon as the game starts, the agent observes a private card that has been dealt to them; this is observation point  $k_1$ , whose set of possible signals is  $S_{k_1} := \{\text{jack, queen, king}\}$ . Should the agent observe the ‘jack’ signal, the decision problem transitions to the decision point  $j_1$ , where the agent must pick one action from the set  $A_{j_1} := \{\text{check, raise}\}$ . If the agent picks ‘raise’, the decision process terminates; otherwise, if ‘check’ is chosen, the process transitions to observation point  $k_2$ , where the agent will observe whether the opponent checks (at which point the interaction terminates) or raises. In the latter case, the process transitions to decision point  $j_4$ , where the agent picks one action from the set  $A_{j_4} := \{\text{fold, call}\}$ . In either case, after the action has been selected, the interaction terminates.

## D. Experimental Evaluation

**Game instances** We numerically investigate agents learning under the COLS in Kuhn and Leduc poker (Kuhn, 1950; Southey et al., 2005), standard benchmark games from the extensive-form games literature.

**Kuhn poker** The two-player variant of Kuhn poker first appeared in (Kuhn, 1950). In this paper, we use the multiplayer variant, as described by Farina et al. (2018). In a multiplayer Kuhn poker game with  $r$  ranks, a deck with  $r$  unique cards is used. At the beginning of the game, each player pays one chip to the pot (*ante*), and is dealt a single private card (their *hand*). The first player to act can *check* or *bet*, *i.e.*, put an additional chip in the pot. Then, the second player can check or bet after a first player’s check, or fold/call the first player’s bet. If no bet was previously made, the third player can either check or bet, and so on in turn. If a bet is made by a player, each subsequent player needs to decide whether to *fold* or *call* the bet. The betting round if all players check, or if every player has had an opportunity to either

fold or call the bet that was made. The player with the highest card who has not folded wins all the chips in the pot.

**Leduc poker** We use a multiplayer version of the classical Leduc hold'em poker introduced by Southey et al. (2005). We employ game instances of rank 3. The deck consists of three suits with 3 cards each. Our instances are parametric in the maximum number of bets, which in limit hold'em is not necessarily tied to the number of players. As in Kuhn poker, we set a cap on the number of raises to one bet. As the game starts, players pay one chip to the pot. Then, two betting rounds follow. In the first one, a single private card is dealt to each player while in the second round a single board card is revealed. The raise amount is set to 2 and 4 in the first and second round, respectively.

For each game, we consider a 3-player and a 4-player variant. The 3-player Kuhn variant uses a deck with  $r = 12$  ranks. The 4-player variant uses a deck with a reduced number of ranks equal to  $r = 5$  to avoid excessive memory usage.

**CFR and CFR(RM+)** Modern variants of counterfactual regret minimization (CFR) are the current practical state-of-the-art in two-player zero-sum extensive-form game solving. We implemented both the original CFR algorithm by Zinkevich et al. (2007), and a more modern variant (which we denote 'CFR(RM+)') using the Regret Matching Plus regret minimization algorithm at each decision point (Tammelin et al., 2015).

**Discussion of results** We compare the maximum per-player regret cumulated by KOMWU for four different choices of constant learning rate  $\eta^{(t)} = \eta \in \{0.1, 1, 5, 10\}$ , against that cumulated by CFR and CFR(RM+).

We remark that the payoff ranges of these games are not  $[0, 1]$  (i.e., the games have not been normalized). The payoff range of Kuhn poker is 6 for the 3-player variant and 8 for the 4-player variant. The payoff range of Leduc poker is 21 for the 3-player variant and 28 for the 4-player variant. So, a learning rate value of  $\eta = 0.1$  corresponds to a significantly smaller learning rate in the normalized game where the payoffs have been shifted and rescaled to lie within  $[0, 1]$  as required in the statements of Properties 2.2 to 2.4.

Results are shown in Figure 4. In all games, we observe that the maximum per-player regret cumulated by KOMWU plateaus and remains constants, unlike the CFR variants. This behavior is consistent with the near-optimal per-player regret guarantees of KOMWU (Theorem 5.5). In the 3-player variant of Leduc poker, we observe that the largest learning rate we use,  $\eta = 10$ , leads to divergent behavior of the learning dynamics.

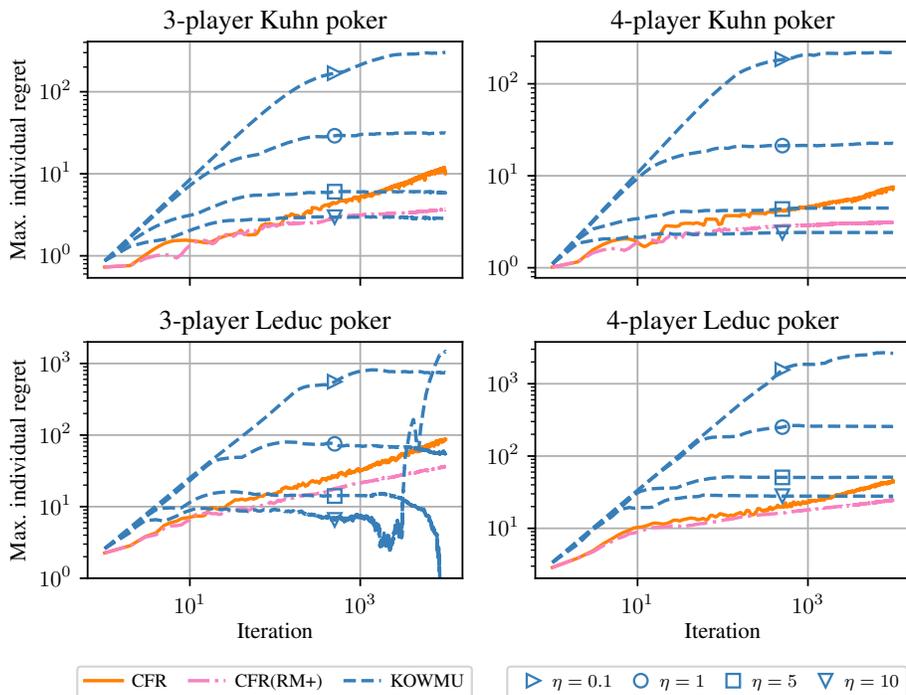


Figure 4. Maximum per-player regret cumulated by KOMWU for four different choices of constant learning rate  $\eta^{(t)} = \eta \in \{0.1, 1, 5, 10\}$ , compared to that cumulated by CFR and CFR(RM+) in two multiplayer poker games.

## E. Proofs

**Theorem 5.2.** For any vectors  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{\Sigma_i}$ , the two following recursive relationships hold:

$$K_{Q_i}(\mathbf{x}, \mathbf{y}) = \mathbf{x}[\emptyset] \mathbf{y}[\emptyset] \prod_{j \in \mathcal{C}_\emptyset} K_j(\mathbf{x}, \mathbf{y}), \quad (13)$$

and, for all decision points  $j \in \mathcal{J}_i$ ,

$$K_j(\mathbf{x}, \mathbf{y}) = \sum_{a \in A_j} \left( \mathbf{x}[ja] \mathbf{y}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{y}) \right). \quad (14)$$

In particular, Equations (13) and (14) give a recursive algorithm to evaluate the polyhedral kernel  $K_{Q_i}$  associated with the sequence-form strategy space of any player  $i$  in an EFG in linear time in the number of sequences  $|\Sigma_i|$ .

*Proof.* In the proof of this result, we will make use of the following additional notation. Given any  $\mathbf{x} \in \mathbb{R}^{\Sigma_i}$  and a  $j \in \mathcal{J}_i$ , we let  $\mathbf{x}_{(j)} \in \mathbb{R}^{\Sigma_{i,j}^*}$  denote the subvector obtained from  $\mathbf{x}$  by only considering sequences  $\sigma \in \Sigma_{i,j}^*$ , that is, the vector whose entries are defined as  $\mathbf{x}_{(j)}[\sigma] = \mathbf{x}[\sigma]$  for all  $\sigma \in \Sigma_{i,j}^*$ .

**Proof of (13)** Direct inspection of the definitions of  $\Pi_i$  and  $\Pi_{i,j}$  (given in Section 5.1), together with the observation that the  $\{\Sigma_{i,j}^* : j \in \mathcal{C}_\emptyset\}$  form a partition of  $\Sigma_i^*$ , reveals that

$$\Pi_i = \left\{ \boldsymbol{\pi} \in \{0, 1\}^{\Sigma_i} : \begin{array}{l} \textcircled{1} \boldsymbol{\pi}[\emptyset] = 1 \\ \textcircled{2} \boldsymbol{\pi}_{(j)} \in \Pi_{i,j} \quad \forall j \in \mathcal{C}_\emptyset \end{array} \right\} \quad (15)$$

The observation above can be summarized informally into the statement that “ $\Pi_i$  is equal, up to permutation of indices, to the Cartesian product  $\times_{j \in \mathcal{C}_\emptyset} \Pi_{i,j}$ ”. The idea for the proof is then to use that Cartesian product structure in the definition of 0/1-polyhedral kernel (4), as follows

$$\begin{aligned} K_{Q_i}(\mathbf{x}, \mathbf{y}) &= \sum_{\boldsymbol{\pi} \in \Pi_i} \prod_{\sigma \in \boldsymbol{\pi}} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \\ &= \sum_{\boldsymbol{\pi} \in \Pi_i} \left( \mathbf{x}[\emptyset] \mathbf{y}[\emptyset] \prod_{j' \in \mathcal{C}_\emptyset} \prod_{\sigma \in \boldsymbol{\pi}_{(j')}} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \right) \\ &= \sum_{\boldsymbol{\pi}_{(j)} \in \Pi_{i,j} \quad \forall j \in \mathcal{C}_\emptyset} \left( \mathbf{x}[\emptyset] \mathbf{y}[\emptyset] \prod_{j' \in \mathcal{C}_\emptyset} \prod_{\sigma \in \boldsymbol{\pi}_{(j')}} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \right) \\ &= \mathbf{x}[\emptyset] \mathbf{y}[\emptyset] \sum_{\boldsymbol{\pi}_{(j)} \in \Pi_{i,j} \quad \forall j \in \mathcal{C}_\emptyset} \left( \prod_{j' \in \mathcal{C}_\emptyset} \prod_{\sigma \in \boldsymbol{\pi}_{(j')}} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \right) \\ &= \mathbf{x}[\emptyset] \mathbf{y}[\emptyset] \prod_{j \in \mathcal{C}_\emptyset} \sum_{\boldsymbol{\pi}_{(j)} \in \Pi_{i,j}} \prod_{\sigma \in \boldsymbol{\pi}_{(j)}} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \\ &= \mathbf{x}[\emptyset] \mathbf{y}[\emptyset] \prod_{j \in \mathcal{C}_\emptyset} K_j(\mathbf{x}, \mathbf{y}), \end{aligned}$$

where the second equality follows from the fact that  $\{\emptyset\} \cup \{\Sigma_{i,j} : j \in \mathcal{C}_\emptyset\}$  form a partition of  $\Sigma_i$ , the third equality follows from (15), the fifth equality from the fact that each  $\boldsymbol{\pi}_{(j)} \in \Pi_{i,j}$  can be chosen independently, and the last equality from the definition of partial kernel function (12).

**Proof of (14)** Similarly to what we did for (13), we start by giving an inductive characterization of  $\Pi_{i,j}$  as a function of the children  $\Pi_{i,j'}$  for  $j' \in \cup_{a \in A_j} \mathcal{C}_{ja}$ . Specifically, direct inspection of the definitions of  $\Pi_{i,j}$ , together with the observation that the  $\{\Sigma_{i,j'}^* : j' \in \cup_{a \in A_j} \mathcal{C}_{ja}\}$  form a partition of  $\Sigma_{i,j}^*$ , reveals that

$$\Pi_{i,j} = \left\{ \boldsymbol{\pi} \in \{0, 1\}^{\Sigma_{i,j}^*} : \begin{array}{l} \textcircled{1} \sum_{a \in A_j} \boldsymbol{\pi}[ja] = 1 \\ \textcircled{2} \boldsymbol{\pi}_{(j')} \in \boldsymbol{\pi}[ja] \cdot \Pi_{i,j'} \quad \forall a \in A_j, j' \in \mathcal{C}_{ja} \end{array} \right\}. \quad (16)$$

From constraint ① together with the fact that  $\pi[ja] \in \{0, 1\}$  for all  $a \in A_j$ , we conclude that exactly one  $a^* \in A_j$  is such that  $\pi[ja^*] = 1$ , while  $\pi[ja] = 0$  for all other  $a \in A_j, a \neq a^*$ . So, we can rewrite (16) as

$$\Pi_{i,j} = \bigcup_{a^* \in A_j} \left\{ \pi \in \{0, 1\}^{\Sigma_{i,j}^*} : \begin{array}{ll} \textcircled{1} \pi[ja^*] = 1 & \\ \textcircled{2} \pi[ja] = 0 & \forall a \in A_j, a \neq a^* \\ \textcircled{3} \pi_{(j')} \in \Pi_{i,j'} & \forall j' \in \mathcal{C}_{ja^*} \\ \textcircled{4} \pi_{(j')} = \mathbf{0} & \forall j' \in \bigcup_{a \in A_j, a \neq a^*} \mathcal{C}_{ja} \end{array} \right\}, \quad (17)$$

where the union is clearly disjoint. The above equality can be summarized informally into the statement that “ $\Pi_{i,j}$  is equal, up to permutation of indices, to a disjoint union over actions  $a^* \in A_j$  of Cartesian products  $\times_{j \in \mathcal{C}_{ja^*}} \Pi_{i,j}$ ”. We can then use the same set of manipulations we already used in the proof of (13) to obtain

$$\begin{aligned} K_j(\mathbf{x}, \mathbf{y}) &= \sum_{\pi \in \Pi_{i,j}} \prod_{\sigma \in \pi} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \\ &= \sum_{\pi \in \Pi_{i,j}} \left( \mathbf{x}[ja^*] \mathbf{y}[ja^*] \prod_{j' \in \mathcal{C}_{ja^*}} \prod_{\sigma \in \pi_{(j')}} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \right) \\ &= \sum_{a^* \in A_j} \sum_{\pi_{j'} \in \Pi_{i,j'} \forall j' \in \mathcal{C}_{ja^*}} \left( \mathbf{x}[ja^*] \mathbf{y}[ja^*] \prod_{j' \in \mathcal{C}_{ja^*}} \prod_{\sigma \in \pi_{(j')}} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \right) \\ &= \sum_{a^* \in A_j} \left( \mathbf{x}[ja^*] \mathbf{y}[ja^*] \prod_{j' \in \mathcal{C}_{ja^*}} \sum_{\pi_{(j')} \in \Pi_{i,j'}} \prod_{\sigma \in \pi_{(j')}} \mathbf{x}[\sigma] \mathbf{y}[\sigma] \right) \\ &= \sum_{a \in A_j} \left( \mathbf{x}[ja] \mathbf{y}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{y}) \right), \end{aligned}$$

where the second equality follows from the fact that the  $\{\Sigma_{i,j'}^* : j' \in \bigcup_{a \in A_j} \mathcal{C}_{ja}\}$  form a partition of  $\Sigma_{i,j}^*$ , third equality follows from (17), the fourth equality from the fact that each  $\pi_{j'} \in \Pi_{i,j'}$  can be picked independently, and the last equality from the definition of partial kernel function (12) as well as renaming  $a^*$  into  $a$ .  $\square$

**Proposition 5.3.** For any player  $i \in [m]$ , vector  $\mathbf{x} \in \mathbb{R}_{>0}^{\Sigma_i}$ , and sequence  $ja \in \Sigma_i^*$ ,

$$\frac{1 - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{ja}) / K_{Q_i}(\mathbf{x}, \mathbf{1})}{1 - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{p_j}) / K_{Q_i}(\mathbf{x}, \mathbf{1})} = \frac{\mathbf{x}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{1})}{K_j(\mathbf{x}, \mathbf{1})}.$$

*Proof.* Note that since  $\mathbf{x} > \mathbf{0}$ , clearly  $K_{Q_i}(\mathbf{x}, \mathbf{1}), K_j(\mathbf{x}, \mathbf{1}) > 0$ . Furthermore, from (11) we have that for all  $\sigma \in \Sigma_i$

$$\begin{aligned} K_{Q_i}(\mathbf{x}, \mathbf{1}) - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_\sigma) &= \langle \phi_{Q_i}(\mathbf{1}) - \phi_{Q_i}(\bar{\mathbf{e}}_\sigma), \phi_{Q_i}(\mathbf{x}) \rangle \\ &= \sum_{\substack{\pi \in \Pi_i \\ \pi[\sigma]=1}} \prod_{\sigma' \in \pi} \mathbf{x}[\sigma'] \\ &> 0. \end{aligned} \quad (18)$$

The above inequality immediately implies that  $0 < K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{p_j}) / K_{Q_i}(\mathbf{x}, \mathbf{1}) < 1$  and therefore all denominators in the statement are nonzero, making the statement well-formed.

In light of (18), we further have

$$\begin{aligned}
 \frac{1 - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{ja})/K_{Q_i}(\mathbf{x}, \mathbf{1})}{1 - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{pj})/K_{Q_i}(\mathbf{x}, \mathbf{1})} &= \frac{\mathbf{x}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{1})}{K_j(\mathbf{x}, \mathbf{1})} \\
 \iff \frac{K_{Q_i}(\mathbf{x}, \mathbf{1}) - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{ja})}{K_{Q_i}(\mathbf{x}, \mathbf{1}) - K_{Q_i}(\mathbf{x}, \bar{\mathbf{e}}_{pj})} &= \frac{\mathbf{x}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{1})}{K_j(\mathbf{x}, \mathbf{1})} \\
 \iff \frac{\sum_{\pi \in \Pi_i, \pi[ja]=1} \prod_{\sigma \in \pi} \mathbf{x}[\sigma]}{\sum_{\pi \in \Pi_i, \pi[pj]=1} \prod_{\sigma \in \pi} \mathbf{x}[\sigma]} &= \frac{\mathbf{x}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{1})}{K_j(\mathbf{x}, \mathbf{1})} \tag{19}
 \end{aligned}$$

We now prove (19). Let

$$\mathcal{A} := \{\pi \in \Pi_i : \pi[ja] = 1\}, \quad \mathcal{B} := \{\pi \in \Pi_i : \pi[pj] = 1\}$$

be the domains of the summations. From the definition of  $\Pi_i$  (specifically, constraints ② in the definition of  $Q_i$ , of which  $\Pi_i$  is a subset; see Section 5.1), it is clear that  $\mathcal{A} \subseteq \mathcal{B}$ . Furthermore, it is straightforward to check, using the definitions of  $\Pi_{i,j}$ ,  $\Pi_i$ , and  $\mathcal{B}$ , that

$$\pi_{(j)} \in \Pi_{i,j} \quad \forall \pi \in \mathcal{B} \tag{20}$$

We now introduce the function  $((\cdot | \cdot)) : \mathcal{B} \times \Pi_{i,j} \rightarrow \mathcal{B}$  defined as follows. Given any  $\pi \in \mathcal{B}$  and  $\pi' \in \Pi_{i,j}$ ,  $((\pi | \pi'))$  is the vector obtained from  $\pi$  by replacing all sequences at or below decision point  $j$  with what is prescribed by  $\pi'$ ; formally,

$$((\pi | \pi'))[\sigma] := \begin{cases} \pi'[\sigma] & \text{if } \sigma \in \Sigma_{i,j}^* \\ \pi[\sigma] & \text{otherwise.} \end{cases} \quad \forall \pi \in \mathcal{B}, \pi' \in \Pi_{i,j} \tag{21}$$

It is immediate to check that  $((\pi | \pi'))$  is indeed an element of  $\mathcal{B}$ . We now introduce the following result.

**Lemma E.1.** *There exists a set  $\mathcal{P} \subseteq \mathcal{B}$  such that every  $\pi'' \in \mathcal{B}$  can be uniquely written as  $\pi'' = ((\pi | \pi'))$  for some  $\pi \in \mathcal{P}$  and  $\pi' \in \Pi_{i,j}$ . Vice versa, given any  $\pi \in \mathcal{P}$  and  $\pi' \in \Pi_{i,j}$ , then  $((\pi | \pi')) \in \mathcal{B}$ .*

*Proof.* The second part of the statement is straightforward. We now prove the first part.

Fix any  $\pi^* \in \Pi_{i,j}$  and let  $\mathcal{P} := \{((\pi | \pi^*)) : \pi \in \mathcal{B}\}$ . It is straightforward to verify that for any  $\pi'' \in \mathcal{B}$ , the choices  $\pi := ((\pi'' | \pi^*)) \in \mathcal{P}$  and  $\pi' := \pi_{(j)} \in \Pi_{i,j}$  satisfy the equality  $((\pi | \pi')) = \pi''$ . So, every  $\pi'' \in \mathcal{B}$  can be expressed in at least one way as  $\pi'' = ((\pi | \pi'))$  for some  $\pi \in \mathcal{P}$  and  $\pi' \in \Pi_{i,j}$ . We now show that the choice above is in fact the unique choice. First, it is clear from the definition of  $((\cdot | \cdot))$  that  $\pi'$  must satisfy  $\pi' = \pi''_{(j)}$ , and so it is uniquely determined. Suppose now that there exist  $\pi, \tilde{\pi} \in \mathcal{P}$  such that  $((\pi | \pi')) = ((\tilde{\pi} | \pi'))$ . Then,  $\pi$  and  $\tilde{\pi}$  must coincide on all  $\sigma \in \Sigma_i \setminus \Sigma_{i,j}^*$ . However, since all elements of  $\mathcal{P}$  are of the form  $((\mathbf{b} | \pi^*))$  for some  $\mathbf{b} \in \mathcal{B}$ , then  $\pi$  and  $\tilde{\pi}$  must also coincide on all  $\sigma \in \Sigma_{i,j}^*$ . So,  $\pi$  and  $\tilde{\pi}$  coincide on all coordinates  $\sigma \in \Sigma_i$ , and the statement follows.  $\square$

Lemma E.1 exposes a convenient combinatorial structure of the set  $\mathcal{B}$ . In particular, it enables us to rewrite the denominator

on the left-hand side of (19) as follows

$$\begin{aligned}
 \sum_{\pi \in \mathcal{B}} \prod_{\sigma \in \pi} x[\sigma] &= \sum_{\pi' \in \mathcal{P}} \sum_{\pi'' \in \Pi_{i,j}} \prod_{\sigma \in ((\pi' | \pi''))} x[\sigma] \\
 &= \sum_{\pi' \in \mathcal{P}} \sum_{\pi'' \in \Pi_{i,j}} \left( \prod_{\substack{\sigma \in ((\pi' | \pi'')) \\ \sigma \in \Sigma_{i,j}}} x[\sigma] \right) \left( \prod_{\substack{\sigma \in ((\pi' | \pi'')) \\ \sigma \notin \Sigma_{i,j}}} x[\sigma] \right) \\
 &= \sum_{\pi' \in \mathcal{P}} \sum_{\pi'' \in \Pi_{i,j}} \left( \prod_{\sigma \in \pi''} x[\sigma] \right) \left( \prod_{\substack{\sigma \in \pi' \\ \sigma \notin \Sigma_{i,j}}} x[\sigma] \right) \\
 &= \left( \sum_{\pi'' \in \Pi_{i,j}} \prod_{\sigma \in \pi''} x[\sigma] \right) \left( \sum_{\substack{\pi' \in \mathcal{P} \\ \sigma \in \pi' \\ \sigma \notin \Sigma_{i,j}}} \prod_{\sigma} x[\sigma] \right) \\
 &= K_j(\mathbf{x}, \mathbf{1}) \cdot \left( \sum_{\substack{\pi' \in \mathcal{P} \\ \sigma \in \pi' \\ \sigma \notin \Sigma_{i,j}}} \prod_{\sigma} x[\sigma] \right), \tag{22}
 \end{aligned}$$

where we used (21) in the third equality.

We can use a similar technique to express the numerator of the left-hand side of (19). Let

$$\Pi_{i,ja} := \{\pi \in \Pi_{i,j} : \pi[ja] = 1\}.$$

Using the constraints that define  $\Pi_i$  and the definition of  $\mathcal{A}$ , it follows immediately that for any  $\pi \in \mathcal{A}$ ,  $\pi_{(j)} \in \Pi_{i,ja}$ . Furthermore, a direct consequence of Lemma E.1 is the following:

**Corollary E.2.** *The same set  $\mathcal{P} \subseteq \mathcal{B}$  introduced in Lemma E.1 is such that every  $\pi'' \in \mathcal{A}$  can be uniquely written as  $\pi'' = ((\pi | \pi'))$  for some  $\pi \in \mathcal{P}$  and  $\pi' \in \Pi_{i,ja}$ .*

Using Corollary E.2 and following the same steps that led to (22), we express the numerator of the left-hand side of (19) as

$$\begin{aligned}
 \sum_{\pi \in \mathcal{A}} \prod_{\sigma \in \pi} x[\sigma] &= \sum_{\pi' \in \mathcal{P}} \sum_{\pi'' \in \Pi_{i,ja}} \prod_{\sigma \in ((\pi' | \pi''))} x[\sigma] \\
 &= \sum_{\pi' \in \mathcal{P}} \sum_{\pi'' \in \Pi_{i,ja}} \left( \prod_{\substack{\sigma \in ((\pi' | \pi'')) \\ \sigma \in \Sigma_{i,j}}} x[\sigma] \right) \left( \prod_{\substack{\sigma \in ((\pi' | \pi'')) \\ \sigma \notin \Sigma_{i,j}}} x[\sigma] \right) \\
 &= \sum_{\pi' \in \mathcal{P}} \sum_{\pi'' \in \Pi_{i,ja}} \left( \prod_{\sigma \in \pi''} x[\sigma] \right) \left( \prod_{\substack{\sigma \in \pi' \\ \sigma \notin \Sigma_{i,j}}} x[\sigma] \right) \\
 &= \left( \sum_{\pi'' \in \Pi_{i,ja}} \prod_{\sigma \in \pi''} x[\sigma] \right) \left( \sum_{\substack{\pi' \in \mathcal{P} \\ \sigma \in \pi' \\ \sigma \notin \Sigma_{i,j}}} \prod_{\sigma} x[\sigma] \right). \tag{23}
 \end{aligned}$$

The statement then follows immediately if we can prove that

$$\sum_{\pi \in \Pi_{i,ja}} \prod_{\sigma \in \pi} x[\sigma] = x[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{1}).$$

To do so, we use the same approach as in the proof of Theorem 5.2. In fact, we can directly use the inductive characterization of  $\Pi_{i,j}$  obtained in (17) to write

$$\Pi_{i,ja} = \left\{ \boldsymbol{\pi} \in \{0,1\}^{\Sigma_{i,j}^*} : \begin{array}{ll} \textcircled{1} \boldsymbol{\pi}[ja] = 1 & \\ \textcircled{2} \boldsymbol{\pi}[ja'] = 0 & \forall a' \in A_j, a' \neq a \\ \textcircled{3} \boldsymbol{\pi}_{(j')} \in \Pi_{i,j'} & \forall j' \in \mathcal{C}_{ja} \\ \textcircled{4} \boldsymbol{\pi}_{(j')} = \mathbf{0} & \forall j' \in \cup_{a' \in A_j, a' \neq a} \mathcal{C}_{ja'} \end{array} \right\},$$

which fundamentally uncovers the *Cartesian-product structure* of  $\Pi_{i,ja}$ . Using the same technique as Theorem 5.2, we then have

$$\begin{aligned} \sum_{\boldsymbol{\pi} \in \Pi_{i,ja}} \prod_{\sigma \in \boldsymbol{\pi}} \mathbf{x}[\sigma] &= \sum_{\boldsymbol{\pi}_{(j')} \in \Pi_{i,j'} \forall j' \in \mathcal{C}_{ja}} \left( \mathbf{x}[ja] \prod_{j' \in \mathcal{C}_{ja}} \prod_{\sigma \in \boldsymbol{\pi}_{(j')}} \mathbf{x}[\sigma] \right) \\ &= \left( \mathbf{x}[ja] \prod_{j' \in \mathcal{C}_{ja}} \sum_{\boldsymbol{\pi}_{(j')} \in \Pi_{i,j'}} \prod_{\sigma \in \boldsymbol{\pi}_{(j')}} \mathbf{x}[\sigma] \right) \\ &= \left( \mathbf{x}[ja] \prod_{j' \in \mathcal{C}_{ja}} K_{j'}(\mathbf{x}, \mathbf{1}) \right), \end{aligned}$$

and the statement is proven.  $\square$

**Proposition 5.1.** *The number of vertices of  $Q_i$  is upper bounded by  $A^{\|Q_i\|_1}$ , where  $A := \max_{j \in \mathcal{J}_i} |A_j|$  is the largest number of possible actions, and  $\|Q_i\|_1 := \max_{\mathbf{q} \in Q_i} \|\mathbf{q}\|_1$ .*

*Proof.* The proof is by induction. As the base case consider a single decision point  $\Delta^b$  with  $b \leq A$  actions. Then the number of vertices is  $b \leq A = A^{\|\Delta^b\|_1}$ .

For the induction step we consider two cases. First, consider a polytope  $Q$  whose root is a decision point with  $b \leq A$  actions, with each action  $a$  leading to a polytope  $Q_a$  whose number of vertices  $v_a$  satisfies the inductive assumption (if some action  $a$  is a terminal action then we overload notation and let  $v_a = 1$  and  $\|Q_a\|_1 = 0$ ). Then, the number of vertices of  $Q$  is

$$\begin{aligned} \sum_{a=1}^b v_a &\leq \sum_{a=1}^b A^{\|Q_a\|_1} \\ &\leq b \cdot A^{\max_{a \in [b]} \|Q_a\|_1} \\ &\leq A \cdot A^{\max_{a \in [b]} \|Q_a\|_1} \\ &= A^{\|Q\|_1}. \end{aligned}$$

Second, consider a polytope  $Q$  whose root is an observation point with  $b$  observations, with each observation  $o$  leading to a polytope  $Q_o$  with  $v_o$  vertices, such that the inductive assumption holds. Then, the number of vertices of  $Q$  is

$$v = \prod_{o=1}^b v_o \leq \prod_{o=1}^b A^{\|Q_o\|_1} \leq A^{\sum_{o=1}^b \|Q_o\|_1} = A^{\|Q\|_1}.$$

$\square$

## F. Further Applications

In this appendix, we illustrate additional 0/1-polyhedral domains in which our polyhedral kernel can be computed efficiently.

**F.1.  $n$ -sets**

We start from  $n$ -sets, that is, the 0/1-polydral set  $\Omega_n^d := \text{co}\{\boldsymbol{\pi} \in \{0, 1\}^d : \|\boldsymbol{\pi}\|_1 = n\}$ . Learning over  $n$ -sets is a classic problem first considered by Warmuth & Kuzmin (2008) with an application to online Principal Component Analysis. They proposed an Online Mirror Descent algorithm operating over the convex hull  $\Omega_n^d$ , with per-iteration complexity of  $\mathcal{O}(d^2)$ . The Follow-the-Perturbed-Leader approach (Kalai & Vempala, 2005) is even faster with per-iteration complexity of  $\mathcal{O}(d \log d)$ , but it often leads to sub-optimal regret bounds (see discussions in (Koolen et al., 2010)). Simulating MWU over the vertices of  $\Omega_n^d$  has been considered in for example (Cesa-Bianchi & Lugosi, 2012), where they proposed to use the general approach of (Takimoto & Warmuth, 2003) to implement this algorithm, leading to per-iteration complexity of  $\mathcal{O}(d^2 n)$ . Below, we show that our kernelized approach admits an even faster per-iteration complexity of  $\mathcal{O}(d \min\{n, d - n\})$ .

**F.1.1. POLYNOMIAL,  $\mathcal{O}(d \min\{n, d - n\})$ -TIME KERNEL EVALUATION**

Let  $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ , and assume for now  $n \leq d - n$ . Introduce the polynomial  $p_{\boldsymbol{x}, \boldsymbol{y}}(z)$  of  $z$ , defined as

$$p_{\boldsymbol{x}, \boldsymbol{y}}(z) := (\boldsymbol{x}[1]\boldsymbol{y}[1]z + 1) \cdots (\boldsymbol{x}[d]\boldsymbol{y}[d]z + 1).$$

It is immediate to see that the coefficient of  $z^n$  in the expansion of  $p_{\boldsymbol{x}, \boldsymbol{y}}(z)$  is exactly  $K_{\Omega_n^d}(\boldsymbol{x}, \boldsymbol{y})$ . Such coefficient can be computed by directly carrying out the multiplication of the binomial terms, keeping track of the term of degree  $0, \dots, n$ . So, each evaluation of  $K_{\Omega_n^d}(\boldsymbol{x}, \boldsymbol{y})$  can be carried out in  $\mathcal{O}(nd)$  time under the assumption that  $n < d - n$ .

If on the other hand  $n < d - n$ , we can repeat the whole argument above for the polynomial  $q_{\boldsymbol{x}, \boldsymbol{y}}(z) := (z + \boldsymbol{x}[1]\boldsymbol{y}[1]) \cdots (z + \boldsymbol{x}[d]\boldsymbol{y}[d])$  instead. In that case, we are interested in the coefficients of  $z^{d-n}$ , which can be computed in  $\mathcal{O}(d(d - n))$  using the same procedure described above.

Putting together the two cases, we conclude that the computation of  $K_{\Omega_n^d}(\boldsymbol{x}, \boldsymbol{y})$  requires  $\mathcal{O}(d \min\{n, d - n\})$  time.

**F.1.2. IMPLEMENTING KOMWU WITH  $\mathcal{O}(d \min\{n, d - n\})$  PER-ITERATION COMPLEXITY**

The result described in the previous paragraph immediately implies that KOMWU can be implemented with  $\mathcal{O}(d^2 \min\{n, d - n\})$ -time iterations. In this subsection we refine that result by showing that it is possible to compute the  $d$  kernel evaluations  $\{K_{\Omega_n^d}(\boldsymbol{x}, \bar{\boldsymbol{e}}_k) : k = 1, \dots, d\}$  required at every iteration by KOMWU so that they take cumulative  $\mathcal{O}(d \cdot \min\{n, d - n\})$  time.

To do so, we build on the technique described in the previous subsection. Assume again that  $n \leq d - n$ . The key insight is that the coefficient of  $z^n$  of the polynomial  $p_{\boldsymbol{x}, \mathbf{1}}(z)/(\boldsymbol{x}[j]z + 1)$  is exactly  $K_{\Omega_n^d}(\boldsymbol{x}, \bar{\boldsymbol{e}}_j)$ . So, to compute all  $\{K_{\Omega_n^d}(\boldsymbol{x}, \bar{\boldsymbol{e}}_k) : k = 1, \dots, d\}$  we can do the following:

1. First, for all  $k = 0, \dots, d$  and  $h = 0, \dots, n$ , we compute the coefficient  $A[k, h]$  of the  $z^h$  in the expansion of  $(\boldsymbol{x}[1]z + 1) \cdots (\boldsymbol{x}[k]z + 1)$

We can compute all such values in  $\mathcal{O}(dn)$  time by using dynamic programming. In particular, we have

$$A[k, h] = \begin{cases} 1 & \text{if } h = 0 \\ 0 & \text{if } k = 0 \wedge h \neq 0 \\ A[k - 1, h] + \boldsymbol{x}[k] \cdot A[k - 1, h - 1] & \text{otherwise.} \end{cases}$$

2. Then, for all  $k = 1, \dots, d + 1$  and  $h = 0, \dots, n$ , we compute the coefficient  $B[k, h]$  of  $z^h$  in the expansion of  $(\boldsymbol{x}[k]z + 1) \cdots (\boldsymbol{x}[d]z + 1)$

Again, we can do that in  $\mathcal{O}(dn)$  time by using dynamic programming. Specifically,

$$B[k, h] = \begin{cases} 1 & \text{if } h = 0 \\ 0 & \text{if } k = d + 1 \wedge h \neq 0 \\ B[k + 1, h] + \boldsymbol{x}[k] \cdot B[k + 1, h - 1] & \text{otherwise.} \end{cases}$$

3. (Note that at this point,  $K_{\Omega_n^d}(\boldsymbol{x}, \mathbf{1})$  is simply  $A[d, n]$ .)
4. For each  $k = 1, \dots, d$ ,  $K_{\Omega_n^d}(\boldsymbol{x}, \bar{\boldsymbol{e}}_j)$  can be computed as

$$K_{\Omega_n^d}(\boldsymbol{x}, \bar{\boldsymbol{e}}_k) = \sum_{h=0}^n A[k - 1, h] \cdot B[k + 1, n - h].$$

The above formula takes  $\mathcal{O}(n)$  time to be computed (we need to iterate over  $h = 0, \dots, n$ ), and we need to evaluate it  $d$  times (once per each  $k = 1, \dots, d$ ). So, computing all  $\{K_{\Omega_n^d}(\mathbf{x}, \bar{\mathbf{e}}_k) : k = 1, \dots, d\}$  takes cumulative  $\mathcal{O}(dn)$  time, as we wanted to show.

As in the previous subsection, the case  $n > d - n$  is symmetric. In that case, the set of values  $\{K_{\Omega_n^d}(\mathbf{x}, \bar{\mathbf{e}}_k) : k = 1, \dots, d\}$  can be computed in cumulative  $\mathcal{O}(d(d - n))$  time.

## F.2. Unit Hypercube

Consider the hypercube  $[0, 1]^d$ , whose vertices are all the vectors in  $\{0, 1\}^d$ . In this case, the polyhedral kernel is simply

$$K_{[0,1]^d}(\mathbf{x}, \mathbf{y}) = (\mathbf{x}[1] \cdot \mathbf{y}[1] + 1) \cdots (\mathbf{x}[d] \cdot \mathbf{y}[d] + 1),$$

which can be clearly evaluated in  $\mathcal{O}(d)$  time. Similarly to  $n$ -sets (Appendix F.1), we can avoid paying an extra  $d$  factor in the per-iteration complexity of KOMWU by using the following procedure:

1. For each  $k = 0, \dots, d$  define  $A[k] := (\mathbf{x}[1] \cdot \mathbf{y}[1] + 1) \cdots (\mathbf{x}[k] \cdot \mathbf{y}[k] + 1)$ . Clearly, the  $A[k]$  values can be computed in  $\mathcal{O}(d)$  cumulative time.
2. For each  $k = 1, \dots, d + 1$ , define  $B[k] := (\mathbf{x}[k] \cdot \mathbf{y}[k] + 1) \cdots (\mathbf{x}[d] \cdot \mathbf{y}[d] + 1)$ . Again, all  $B[k]$  values can be computed in  $\mathcal{O}(d)$  cumulative time.
3. For each  $k = 1, \dots, d$ , we have that  $K_{[0,1]^d}(\mathbf{x}, \bar{\mathbf{e}}_k) = A[k - 1] \cdot B[k + 1]$ . Hence, we can compute  $\{K_{[0,1]^d}(\mathbf{x}, \bar{\mathbf{e}}_k) : k = 1, \dots, d\}$  in cumulative  $\mathcal{O}(d)$  time.

## F.3. Flows in Directed Acyclic Graphs

The polytope  $\mathcal{F}$  of flows in a generic directed acyclic graphs (DAGs) has vertices with 0/1 integer coordinates, corresponding to paths in the DAG. The 0/1-polyhedral kernel  $K_{\mathcal{F}}$  corresponding to the set of flows in a DAG coincides with the kernel function introduced by Takimoto & Warmuth (2003), which was shown to be computable in polynomial-time in the size of the DAG. Consequently,  $K_{\mathcal{F}}$  admits polynomial-time (in the size of the DAG) evaluation.

## F.4. Permutations

When  $\mathcal{P}$  is the convex hull of the set of all  $d \times d$  permutation matrices, it is believed that  $K_{\mathcal{P}}$  cannot be evaluated in polynomial time in  $\mathcal{O}(d)$ , since the computation of the permanent of a matrix  $\mathbf{A}$  can be expressed as  $K_{\Omega}(\mathbf{A}, \mathbf{1})$ . However, an  $\epsilon$ -approximate computation of  $K_{\mathcal{P}}$  can be performed in  $\mathcal{O}(\text{poly}(d, \log(1/\epsilon)))$  for any  $\epsilon > 0$  by using a landmark result by Jerrum et al. (2004). We refer the interested reader to the paper by Cesa-Bianchi & Lugosi (2012, Section 5.3).

## F.5. Cartesian Product

Finally, we remark that when two 0/1-polyhedral sets have efficiently-computable 0/1-polyhedral kernels, then so does their Cartesian product. Specifically, let  $\Omega \subseteq \mathbb{R}^d, \Omega' \subseteq \mathbb{R}^{d'}$  be 0/1-polyhedral sets, and let  $K_{\Omega}, K_{\Omega'}$  be their corresponding 0/1-polyhedral kernels. Then, it follows immediately from the definition that the polyhedral kernel of  $\Omega \times \Omega'$  satisfies

$$K_{\Omega \times \Omega'}\left(\begin{pmatrix} \mathbf{x} \\ \mathbf{x}' \end{pmatrix}, \begin{pmatrix} \mathbf{y} \\ \mathbf{y}' \end{pmatrix}\right) = K_{\Omega}(\mathbf{x}, \mathbf{y}) \cdot K_{\Omega'}(\mathbf{x}', \mathbf{y}') \quad \forall \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}, \begin{pmatrix} \mathbf{x}' \\ \mathbf{y}' \end{pmatrix} \in \mathbb{R}^d \times \mathbb{R}^{d'}.$$