MIT 6.S890 — Topics in Multiagent Learning

Lecture 14

Combinatorial games and Kernelized MWU

Instructor: Prof. Gabriele Farina

Extensive-form games belong to a larger class of games called *combinatorial games*. Combinatorial games are characterized by the following properties:

- For every player i, the set of deterministic strategies can be represented as a set $\mathcal{V}_i$ of bit strings, that is, elements in $\{0,1\}^{d_i}$ for some appropriate dimension d_i; and
- The utility of every player is a multilinear function of the strategies.

1 Examples of Combinatorial Games

Both normal-form games and extensive-form games, the classes of games we have been studying so far, are examples of combinatorial games. In this section, we also show that several other important classes of games are combinatorial games.

1.1 Normal-form games

In normal-form games, we already know that the utility of each player is multilinear in the distributions over actions used by the players. The deterministic strategies correspond to deterministic distirbutions, which have the form

$$\begin{pmatrix}1\\0\\0\\\vdots\\0\end{pmatrix}, \quad \begin{pmatrix}0\\1\\0\\\vdots\\0\end{pmatrix}, \quad \begin{pmatrix}0\\0\\1\\\vdots\\0\end{pmatrix}, \quad ..., \quad \begin{pmatrix}0\\0\\0\\\vdots\\1\end{pmatrix}.$$

Hence, normal-form games are combinatorial games.

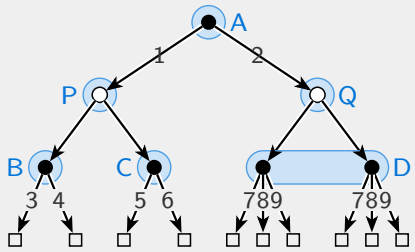
Remark 1.1. The number of deterministic strategies in a normal-form game is $|A_i|$, where A_i is the set of actions of the player.

1.2 Extensive-form games

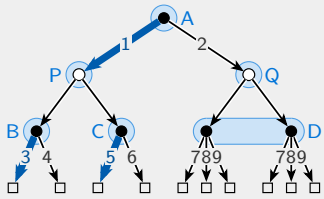
Perhaps slightly less obvious is that extensive-form games are combinatorial games. There, we already know that the utility of each player is multilinear when using the sequence form representation of strategies. Deterministic strategies for the tree in the sequence-form representation contain entries in $\{0,1\}$. As a reminder, these strategies correspond to the set of all determinstic contingency plans for the entire tree. We illustrate this with an example.

⋆These notes are class material that has not undergone formal peer review. The TA and I are grateful for any reports of typos.

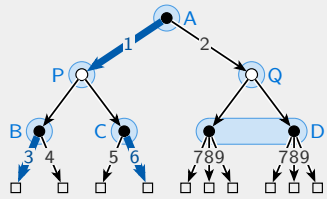
Example 1.1. For example, consider the following two-player game, where black nodes belong to Player 1 and white nodes belong to Player 2.



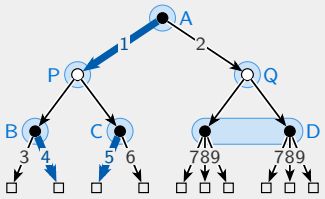
The information set D for Player 1 encodes the fact that Player 1 does not observe Player 2's action at Q. In this small game, the following 7 strategies $v_1, ..., v_7$ form the vertices of the sequence-form polytope.



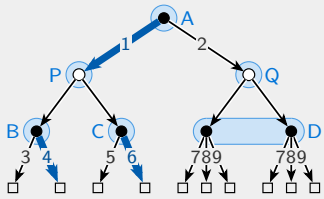
1 2 3 4 5 6 7 8 9
$v_1 := (1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0)$



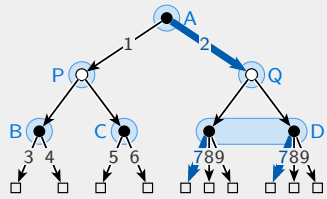
1 2 3 4 5 6 7 8 9
$v_2 := (1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0)$



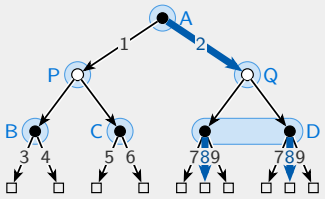
1 2 3 4 5 6 7 8 9
$v_3 := (1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0)$



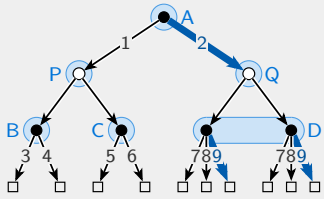
1 2 3 4 5 6 7 8 9
$v_4 := (1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0)$



1 2 3 4 5 6 7 8 9
$v_5 := (0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0)$



1 2 3 4 5 6 7 8 9
$v_6 := (0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0)$



1 2 3 4 5 6 7 8 9
$v_7 := (0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1)$

Remark 1.2. We remark that the number of deterministic strategies in an extensive-form game is in general exponential in the size of the game tree. In particular, we can easily bound the number of deterministic strategies of a generic player i as $|\mathcal{V}_i| \leq A_i^{|I_i|}$, where I_i is the set of information sets of the player and A_i is the maximum number of actions at any of those information sets.

In contrast, the *dimension* d_i of the bit strings is just the *sum* of the number of actions across all information sets of the player.

1.3 Resource allocation games

Another class of combinatorial games goes under the name of *resource allocation games*, of which the game Colonel Blotto is the most famous example. In these games, the set of deterministic strategies is the set of all possible ways to allocate a fixed amount of resources among a set of activities. In Colonel Blotto, the resources are soldiers, and the activities are the battlefields. We can represent these allocations using bit strings that represent a table where on the rows we have activities, and on the columns we have amounts of allocated resources. We can then one-hot encode how many resources we would like to allocate to each activity, making sure that the total number of resources allocated matches the number of resources the player has.

1.4 Games on graphs

Another example of combinatorial games are games on graphs. In these games, the set of deterministic strategies is the set of all possible paths in the graph. Here, we can use bit strings of length equal to the number of edges in the graph to represent the paths. We can then one-hot encode which edges are traversed in the path.

1.5 Games on fixed-size subsets (aka. m-sets)

Finally, there exist games in which the actions correspond to picking a subsets of a given size. In these games, the set of deterministic strategies is encoded as binary strings that one-hot encode which of the elements are selected, that is,

$$\mathcal{V}_i = \{x \in \{0,1\}^d : \|x\|_1 = m\}.$$

The setting above goes under the name of *m-sets* in the machine learning literature.

2 Learning in combinatorial games and kernelization

How can we learn in a combinatorial game? To start, if we had a way to compute *projections* onto the convex hull of $\mathcal{V}_i$ (*i.e.*, the polytope spanned by $\mathcal{V}_i$), we could perform projected gradient descent. However, computing projections onto the convex hull of $\mathcal{V}_i$ might be hard. Furthermore, the regret bounds (and hence equilibrium approximation rates) are typically far from optimal. For example, in normal-form games we have observed that projected gradient descent achieves a regret of $O\left(\sqrt{d_i T}\right)$, while the optimal regret is $O\left(\sqrt{\log(d_i) T}\right)$, an *exponential* improvement in terms of the dimension of the set. We were able to achieve the latter, optimal bound using the multiplicative weights update (MWU) algorithm.

The reasons to like the MWU algorithm extend even beyond the logarithmic dependence on the number of actions. As we saw in Lecture 6, the *optimistic* variant of MWU (OMWU) achieves a very strong dependence on the number of repetitions T, going from a $\sqrt{T}$ dependence to a $O(\text{polylog}(T))$ dependence.

Alas, multiplicative weights is an algorithm that is designed for probability simplices, and it is not clear how to apply it to combinatorial games. In this lecture, we show how to *kernelize* the MWU algorithm to combinatorial games, and how to implement it efficiently in several important classes of combinatorial games.

2.1 Multiplicative weights on the set of deterministic strategies

We can use MWU (or OMWU) in combinatorial games by letting it pick distirbutions over the set of deterministic strategies $\mathcal{V}_i$. In other words, we can run MWU to keep tallies on how each deterministic strategy is performing, and then use these tallies to compute the next distribution over the strategies

(of course, skewing the distribution towards better-performing strategies). We call this approach "Vertex MWU".

Algorithm 1: Vertex MWU/OMWU (Player i)

$\lambda^{(1)} :=$ uniform distribution over $\mathcal{V}_i$

$g^{(0)} := 0 \in \mathbb{R}^{d_i}$

function `NextStrategy`()

  Play mixed strategy $x^{(t)} := \sum_{v \in \mathcal{V}_i} \left(\lambda^{(t)}[v] \cdot v\right)$ // How to do efficiently?

function `ObserveUtility`$\left(g^{(t)}\right)$

  (If optimistic) Perform optimistic correction $\tilde{g}^{(t)} := 2g^{(t)} - g^{(t-1)}$

  (If not optimistic) Let $\tilde{g}^{(t)} := g^{(t)}$

  Update $\lambda^{(t+1)}[v] \propto \lambda^{(t)}[v] \cdot \exp\left\{\eta \cdot \left\langle \tilde{g}^{(t)}, v \right\rangle\right\}$ for all $v \in \mathcal{V}_i$ // How to do efficiently?

Remark 2.1. In the game setting (*i.e.*, what we called the "canonical learning setup" in Lecture 4), the inputs to `ObserveUtility` are the gradients of the utility. These can be always computed in polynomial time in d and n.

2.2 Main result

It is not obvious that we can implement the Vertex MWU/OMWU algorithm (Algorithm 1) efficiently, that is, without paying linear time in $|\mathcal{V}_i|$ when the latter set is exponential in the dimension d_i. Yet, as pointed by Farina, G., Lee, C.-W., Luo, H., & Kroer, C. [Far+22], that is possible in many cases, including all settings mentioned above. The key insight is that we can use a *kernel function* to compute the expectations and the gradients in the algorithm efficiently. Specifically, in this lecture we will prove the following result. For simplicity, we will assume a player has been fixed and drop the subscript i from the notation.

Theorem 2.1 (Main theorem). There exists a *kernel function* $K_{\mathcal{V}} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, which depends on the combinatorial structure of the set $\mathcal{V}$ of the player, such that Algorithm 1 can be implemented using $d + 1$ evaluations of $K_{\mathcal{V}}$ at each iteration.

We call this kernelized implementation of Algorithm 1 the *Kernelized MWU/OMWU algorith*, abbreviated as KMWU or KOMWU depending on whether optimism is used.

Note that Theorem 2.1 is crucially independent on the number of strategies $|\mathcal{V}|$, and only depends on the *dimension* d of the strategy set, which is polynomial in the size of the input game (for example, in extensive-form games, d is upper bounded by the number of edges in the game tree)! The main takeaway is the following.

Corollary 2.1. As long as the kernel function $K_{\mathcal{V}}$ can be evaluated efficiently, then Algorithm 1 can be simulated efficiently too.

2.3 The 0/1-polyhedral kernel

In order to introduce the notion of 0/1-polyhedral kernel, we need to first introduce the notion of 0/1-polyhedral feature map.

Definition 2.1 (0/1-polyhedral feature map). The 0/1-polyhedral feature map $\phi_{\mathcal{V}}$ is defined as

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

As is common with kernels, we define the 0/1-polyhedral kernel as the inner product of the feature maps.

Definition 2.2 (0/1-polyhedral kernel).

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

2.4 Keeping track of the distribution over vertices

We start from showing how the 0/1-polyhedral kernel can help implement Line 8 of Algorithm 1 efficiently. The key insight is that the strategies $\lambda^{(t)}$ are fully captured by the feature map of a low-dimensional vector at all iterations.

Theorem 2.2. At all times t, the distribution $\lambda^{(t)}$ over strategies $\mathcal{V}$ computed by Algorithm 1 is proportional to the feature map of the vector

$$b^{(t)} \coloneqq \exp\left\{ \eta \sum_{\tau=1}^{t-1} \tilde{g}^{(\tau)} \right\}.$$

Proof. We prove the result by induction over the time t.

- Base case. At time $t = 1$, we have

$$b^{(1)} = 1 \qquad \Longrightarrow \qquad \phi_{\mathcal{V}}\big(b^{(1)}\big) = 1 \propto \frac{1}{|\mathcal{V}|} = \lambda^{(1)}.$$

- Inductive step. At time $t+1$, the probability of the strategy $v \in \mathcal{V}$ computed by KOMWU is

$$\lambda^{(t+1)}[v] \propto \lambda^{(t)}[v] \cdot \exp\big\{\eta \big\langle \tilde{g}^{(t)}, v \big\rangle\big\}.$$

  The key insight now is that since $v \in \{0,1\}^d$, then

$$\exp\big\{\eta \big\langle \tilde{g}^{(t)}, v \big\rangle\big\} = \exp\left\{ \eta \sum_{k:v[k]=1} \tilde{g}^{(t)}[k] \right\} = \prod_{k:v[k]=1} \exp\big\{\eta \tilde{g}^{(t)}[k]\big\}.$$

  Substituting the inductive hypothesis,

$$\begin{aligned}
\lambda^{(t+1)}[v] &\propto \phi_{\mathcal{V}}\big(b^{(t)}\big)[v] \cdot \prod_{k:v[k]=1} \exp\big\{\eta \tilde{g}^{(t)}[k]\big\} \\
&= \left(\prod_{k:v[k]=1} b^{(t)}[k] \right) \cdot \left(\prod_{k:v[k]=1} \exp\big\{\eta \tilde{g}^{(t)}[k]\big\} \right) \\
&= \prod_{k:v[k]=1} b^{(t+1)}[k] \\
&= \phi_{\mathcal{V}}\big(b^{(t+1)}\big)[v],
\end{aligned}$$

  completing the proof. □

In fact, we can even slightly refine the previous result by quantifying exactly the proportionality constant. We do so in the next corollary.

Corollary 2.2. At all times t, one has

$$\lambda^{(t)} = \frac{\phi_{\mathcal{V}}(b^{(t)})}{K_{\mathcal{V}}(b^{(t)}, 1)}.$$

Proof. Since we know from Theorem 2.2 that $\lambda^{(t)}[v] \propto \phi_{\mathcal{V}}(b^{(t)})[v]$, and the sum $\sum_{v\in\mathcal{V}} \lambda^{(t)}[v] = 1$, the proportionality constant must be the inverse of

$$\sum_{v\in\mathcal{V}} \phi_{\mathcal{V}}(b^{(t)})[v] = \sum_{v\in\mathcal{V}} \phi_{\mathcal{V}}(b^{(t)})[v] \cdot 1 = \sum_{v\in\mathcal{V}} \phi_{\mathcal{V}}(b^{(t)})[v] \cdot \phi_{\mathcal{V}}(1)[v] = K_{\mathcal{V}}(b^{(t)}, 1),$$

completing the proof. □

2.5 Reconstructing the expectation

We now show how one can reconstruct the expectation

$$\sum_{v\in\mathcal{V}} (\lambda^{(t)}[v] \cdot v)$$

which is needed in `NextStrategy`. The key is provided in the next theorem, which extends a nice insight by Takimoto, E., & Warmuth, M. K. [TW03] .

Theorem 2.3. At all times t, the expectation $\sum_{v\in\mathcal{V}} \lambda^{(t)}[v] \cdot v$ can be computed via $d+1$ kernel computations according to the formula

$$\sum_{v\in\mathcal{V}} (\lambda^{(t)}[v] \cdot v) = \left(1 - \frac{K_{\mathcal{V}}(b^{(t)}, \overline{e}_1)}{K_{\mathcal{V}}(b^{(t)}, 1)}, ..., 1 - \frac{K_{\mathcal{V}}(b^{(t)}, \overline{e}_d)}{K_{\mathcal{V}}(b^{(t)}, 1)}\right),$$

where

$$\overline{e}_k := (1, ..., 1, 0, 1, ..., 1) = 1 - e_k \in \mathbb{R}^d$$

is the vector that is equal to 1 in all coordinates except the i-th one where it is zero.

Proof. The key insight is that, for all $k \in [d]$,

$$\phi_{\mathcal{V}}(\overline{e}_k)[v] = \prod_{j:v[j]=1} \overline{e}_{k[j]} = \begin{cases} 0 \text{ if } v[k] = 1 \\ 1 \text{ otherwise} \end{cases}$$
$$= 1 - v[k].$$

So,

$$(\phi_{\mathcal{V}}(1) - \phi_{\mathcal{V}}(\overline{e}_k))[v] = \phi_{\mathcal{V}}(1)[v] - \phi_{\mathcal{V}}(\overline{e}_k)[v] = 1 - (1 - v[k]) = v[k],$$

and we can write, for all $k \in [d]$,

$$\left(\sum_{v\in\mathcal{V}} \lambda^{(t)}[v] \cdot v\right)[k] = \sum_{v\in\mathcal{V}} (\lambda^{(t)}[v] \cdot (\phi_{\mathcal{V}}(1) - \phi_{\mathcal{V}}(\overline{e}_k))[v])$$
$$= \frac{1}{K_{\mathcal{V}}(b^{(t)}, 1)} \sum_{v\in\mathcal{V}} \phi_{\mathcal{V}}(b^{(t)})[v] \cdot (\phi_{\mathcal{V}}(1)[v] - \phi_{\mathcal{V}}(\overline{e}_k)[v])$$
$$= \frac{1}{K_{\mathcal{V}}(b^{(t)}, 1)} (K_{\mathcal{V}}(b^{(t)}, 1) - K_{\mathcal{V}}(b^{(t)}, \overline{e}_k)) = 1 - \frac{K_{\mathcal{V}}(b^{(t)}, \overline{e}_k)}{K_{\mathcal{V}}(b^{(t)}, 1)},$$

which is the statement. □

3 Examples of efficiently-computable kernels

The previous theorems show that, as long as the kernel can be computed efficiently, then the OMWU algorithm can be simulated efficiently even if the cardinality of every $\mathcal{V}_i$ is exponential in the size of the input game.

3.1 Hypercube

When $\mathcal{V} = \{0,1\}^d$, then

$$K_{\mathcal{V}}(x,y) = (1 + x[1]\, y[1])(1 + x[2]\, y[2]) \cdot \ldots \cdot (1 + x[d]\, y[d])$$

can be evaluated in linear time in d.

3.2 Multiple choices: m-sets

When $\mathcal{V} = \{x \in \{0,1\}^d : \|x\|_1 = m\}$, the kernel can be computed efficiently by using dynamic programming or equivalently, by considering the polynomial of γ given as

$$\Big(1 + \gamma \cdot x[1]y[1]\Big)\Big(1 + \gamma \cdot x[2]y[2]\Big)\cdots\Big(1 + \gamma \cdot x[d]y[d]\Big)$$

When expanding the product (which can be done in polynomial time in d), the coefficient of the term γ^m is the kernel value, since it will be precisely the sum of the products of x, y at all choices of subsets of m coordinates.

3.3 Set of flows in a DAG

In this case, the kernel can be computed in linear time in the number of DAG edgs by performing dynamic programming on the topological order of the DAG. This case was already covered by Takimoto, E., & Warmuth, M. K. [TW03], though we remark that in kernelized OMWU the concept of weight pushing is not needed, so the final algorithms are slightly different.

3.4 Extensive-form games

In particular, as we will see below, KOMWU can be implemented with linear-time iterations in the number of sequences d_i.

■ **Worst-case linear complexity for a single evaluation.** We start by verifying that the sequence-form kernel can be evaluated in linear time for any pair of points $x, y \in \mathbb{R}^{d_i}$. To do so, we introduce a *partial kernel function* $K_I : \mathbb{R}^{d_i} \times \mathbb{R}^{d_i} \to \mathbb{R}$ for every information set $I \in \mathcal{I}$,

$$K_I : \mathbb{R}^{d_i} \times \mathbb{R}^{d_i} \to \mathbb{R}, \qquad K_I(x,y) := \sum_{v \in \mathcal{V}_{\succeq I}} \prod_{k : v[k]=1} x[k]y[k].$$

where $\mathcal{V}_{\succeq I}$ denotes the projection of the strategy set $\mathcal{V}$ onto only those actions at I or below (removing duplicates). We have the following.

Theorem 3.1. For any vectors $x, y \in \mathbb{R}^{d_i}$, the two following recursive relationships hold:

$$K_{\mathcal{V}}(x,y) = \prod_{I \in \mathcal{I}_{\text{top}}} K_I(x,y), \tag{1}$$

where $\mathcal{I}_{\text{top}}$ denotes the "top-level" information sets (those information sets that contain nodes where the player is acting for the first time), and, for all information sets $I \in \mathcal{I}$,

$$K_I(x,y) = \sum_{a \in \mathcal{A}(I)} \left(x[Ia]y[Ia] \prod_{I' \in \mathcal{C}(I)} K_{I'}(x,y) \right), \tag{2}$$

where $\mathcal{C}(I)$ denotes those information sets that are immediate successors of I. In particular, (1) and (2) give a recursive algorithm to evaluate the polyhedral kernel $K_{\mathcal{V}}$ associated with the strategy space of any player i in an imperfect-information extensive-form game in linear time in d_i.

Corollary 3.1. For each player i, KOMWU can be implemented in polynomial time in the size of the game tree. As a consequence, all the regret guarantees of OMWU can be extended to the setting of imperfect-information extensive-form games as a black box.

We also make the following tangential remarks.

Remark 3.1. The feasibility of the kernelization approach in extensive-form games counters the common belief that learning in the normal-form equivalent of an extensive-form game is computationally infeasible.

Remark 3.2. Surprisingly, even for the non-optimistic version the kernelized MWU algorithm achieves better regret bounds than all prior algorithms for learning in extensive-form games.

■ **Amortizated constant-time kernel computation.** We can actually refine the result above by showing an implementation of KOMWU with *linear-time* per-iteration complexity in the size of the game tree, by exploiting the structure of the particular set of kernel evaluations needed at every iteration and amortizing computation of the $d + 1$ kernels required by KOMWU at each iteration.

Bibliography

[Far+22] G. Farina, C.-W. Lee, H. Luo, and C. Kroer, "Kernelized Multiplicative Weights for 0/1-Polyhedral Games: Bridging the Gap Between Learning in Extensive-Form and Normal-Form Games," in *International Conference on Machine Learning*, 2022.

[TW03] E. Takimoto and M. K. Warmuth, "Path kernels and multiplicative updates," *Journal of Machine Learning Research*, vol. 4, pp. 773–818, 2003.

6.S890 Topics in Multiagent Learning

Fall 2024