

Lecture 16

High-dimensional games

Instructor: Prof. Constantinos Daskalakis (costis@mit.edu)

Extensive-form games belong to a larger class of games that we will call *combinatorial games*. Two properties characterize the representation we use:

- For each player i , the deterministic strategies form a nonempty set $V_i \subseteq \{0, 1\}^{d_i}$ of bit strings.
- Each player's utility is a multilinear function of these strategy vectors.

Write $X_i = \text{conv}(V_i)$. A distribution $\lambda_i \in \Delta(V_i)$ induces a mean strategy $\mathbf{x}_i = \sum_{\mathbf{v} \in V_i} \lambda_i(\mathbf{v}) \mathbf{v} \in X_i$. By multilinearity, when the players randomize independently, their expected utilities depend only on these means. Thus we can work in dimension d_i , even when the number of deterministic strategies $|V_i|$ is exponentially larger.

The central question of this lecture is whether [multiplicative weights](#) (Section L5.1.4) can also exploit this compact representation. We follow the kernelization approach of [Farina, G., Lee, C.-W., Luo, H., & Kroer, C. \[Far+22\]](#).

L16.1 Examples of combinatorial games

Normal-form and extensive-form games are both examples. Resource allocation, paths, and fixed-size subsets provide other useful strategy spaces.

L16.1.1 Normal-form games

If Player i has d_i actions, represent action k by the unit vector $\mathbf{e}_k \in \{0, 1\}^{d_i}$. Then

$$V_i = \{\mathbf{e}_1, \dots, \mathbf{e}_{d_i}\}, \quad X_i = \Delta(d_i).$$

The mean of a distribution over these vectors is exactly the usual mixed strategy. Expected utility is multilinear in the players' mixed strategies, so this is a combinatorial-game representation. Here $|V_i| = d_i$: the number of strategies and their encoding dimension coincide.

L16.1.2 Extensive-form games

In a finite perfect-recall extensive-form game, utility is multilinear in the players' [sequence-form strategies](#) (Section L7.3.3). The vertices of a player's sequence-form polytope have

entries in $\{0,1\}$. They describe *reduced pure realization plans*: choices below an unselected own action have realization weight zero. Different full contingency plans can therefore represent the same vertex.

Example L16.1 (Sequence-form vertices). In the game below, black nodes belong to Player 1 and white nodes to Player 2. Player 1 observes the move at P , but not the move at Q : the two nodes in information set D require the same action. The nine numbered actions give the coordinates of Player 1's realization plan.

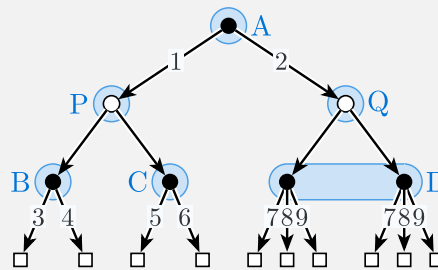


Figure L16.1: An extensive-form game. Repeated labels 7, 8, and 9 belong to one information set and therefore one set of sequence coordinates.

The seven reduced pure realization plans are shown next. Choosing root action 1 leaves two independent binary choices at B and C , giving four vertices. Choosing root action 2 leaves a three-way choice at D , giving three more. Blue edges indicate the coordinates equal to one.

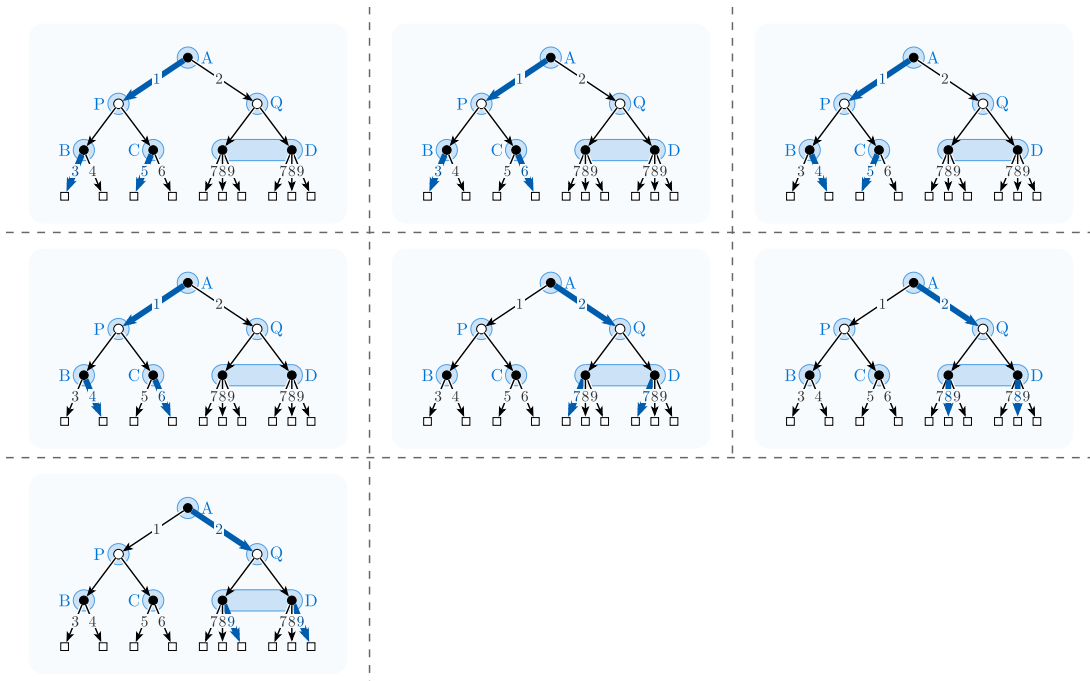


Figure L16.2: The seven binary vertices of Player 1's sequence-form polytope, shown as reduced pure realization plans. See also the [reduced normal-form plans in the modeling notes \(Section L7.3.1\)](#).

Remark L16.2 (Number of strategies versus dimension). If I_i is the set of Player i 's information sets and A_i is the largest number of actions at one of them, then

$$|V_i| \leq \prod_{I \in I_i} |A(I)| \leq A_i^{|I_i|}.$$

In contrast, the encoding dimension is $d_i = \sum_{I \in I_i} |A(I)|$, omitting the empty sequence, whose realization weight is always one. An exponential number of vertices can fit in a representation of size linear in the game tree.

L16.1.3 Resource allocation games

In resource allocation games such as Colonel Blotto, a player distributes an integer budget R among B activities or battlefields. Use a table with one row per battlefield and columns indexed by $s = 0, \dots, R$. Set $v_{j,s} = 1$ exactly when battlefield j receives s units. The valid bit strings satisfy

$$\sum_{s=0}^R v_{j,s} = 1 \quad (j = 1, \dots, B), \quad \sum_{j=1}^B \sum_{s=0}^R s v_{j,s} = R.$$

Thus $d = B(R+1)$. There are $\binom{R+B-1}{B-1}$ possible allocations, which can be much larger than d .

For the usual additive battlefield utilities, each battlefield's payoff depends on the allocations to that battlefield. Its expectation is multilinear in the players' one-hot vectors, and summing over battlefields preserves multilinearity. The representation is polynomial in the budget value R ; it need not be polynomial in the length of a binary encoding of R .

L16.1.4 Games on graphs

A path in a directed acyclic graph can be represented by its edge-indicator vector: $v_e = 1$ exactly when the path uses edge e . Fix a source and destination with at least one connecting path. The dimension is the number of edges, although there may be exponentially many paths.

Edge-additive interaction payoffs can be multilinear in these indicators. For example, the weighted overlap of two players' paths is $\sum_e c_e v_{1,e} v_{2,e}$. We will use the acyclic structure to evaluate the corresponding kernel efficiently.

L16.1.5 Games on fixed-size subsets

If a player selects exactly m of d objects, use the set

$$V = \left\{ v \in \{0, 1\}^d : \sum_{k=1}^d v_k = m \right\}, \quad 0 \leq m \leq d.$$

These strategies are often called m -sets. There are $\binom{d}{m}$ of them. Their indicator vectors again give a compact representation for games with multilinear selection payoffs. If the cardinality restriction is removed, the strategy set is the entire hypercube $\{0, 1\}^d$.

L16.2 Learning in combinatorial games and kernelization

One way to learn over X_i is projected gradient descent, provided projections onto X_i can be computed efficiently. But the choice of learning algorithm also affects regret. For normal-form games with coordinate payoffs in $[-1, 1]$, the standard Euclidean analysis gives an $O(\sqrt{d_i T})$ bound, whereas MWU gives $O(\sqrt{T \log d_i})$. The geometry used by MWU yields a substantially better dependence on the number of actions.

Optimistic MWU (Section S4.1.1) provides another motivation. In suitable self-play settings, optimism improves the dependence on the number of rounds; those guarantees require their own payoff and learning-rate hypotheses. An exact implementation on combinatorial strategies transfers the corresponding normal-form guarantees under the same hypotheses. Here we concentrate on how to obtain that implementation.

L16.2.1 Multiplicative weights on deterministic strategies

Fix one player and drop the player index. Against fixed opponents, write the utility as $u(\mathbf{x}) = \langle \mathbf{g}, \mathbf{x} \rangle$, where $\mathbf{g} \in \mathbb{R}^d$ is its gradient. We assume this gradient can be computed efficiently from the opponents' mean strategies. This is an assumption on the payoff representation, separate from having a compact strategy space.

We can run MWU on the simplex $\Delta(V)$ by maintaining a weight for each deterministic strategy. The vector we play is the expectation of that distribution. We call this algorithm *vertex MWU*; replacing the observed gradient by an optimistic correction gives vertex OMWU.

Algorithm L16.1: Vertex MWU/OMWU

```

1 Initialize  $\lambda_1$  uniformly on  $V$ , set  $\mathbf{g}_0 = 0$ , and set  $t = 1$ .
2 function NextStrategy():
3    $\perp$  Return  $\mathbf{x}_t = \sum_{v \in V} \lambda_t(v) \mathbf{v}$ .
4 function ObserveUtility( $\mathbf{g}_t$ ):
5   Set  $\mathbf{h}_t = \mathbf{g}_t$  (MWU) or  $\mathbf{h}_t = 2\mathbf{g}_t - \mathbf{g}_{t-1}$  (OMWU).
6   For every  $\mathbf{v} \in V$ , set  $\lambda_{t+1}(\mathbf{v}) \propto \lambda_t(\mathbf{v}) \exp(\eta \langle \mathbf{h}_t, \mathbf{v} \rangle)$ .
7    $\perp$  Normalize the weights to sum to one, then increment  $t$ .
```

Algorithm L16.1: Vertex MWU/OMWU. Both the expectation and the weight update appear to require enumerating all vertices.

The regret of the played means is exactly the regret of these distributions over vertices, since $\langle \mathbf{g}_t, \mathbf{x}_t \rangle = \sum_v \lambda_t(v) \langle \mathbf{g}_t, \mathbf{v} \rangle$. The comparator can be any point of X : a linear function attains its maximum at a vertex.

For ordinary MWU with fixed $\eta > 0$ and $|\langle \mathbf{g}_t, \mathbf{v} \rangle| \leq M$ for every $t, \mathbf{v}$, where $M > 0$, the usual exponential-weights analysis gives

$$\text{Reg}_T \leq \frac{\log|V|}{\eta} + \frac{\eta M^2 T}{2}.$$

For $|V| > 1$, choosing $\eta = \sqrt{\frac{2 \log|V|}{M^2 T}}$ gives $\text{Reg}_T \leq M \sqrt{2T \log|V|}$. If $|V| = 1$, regret is zero. This is the ordinary MWU bound; the optimistic update has a separate analysis.

Proof Sketch. Let $W_t = \sum_v \exp(\eta \sum_{\tau < t} \langle \mathbf{g}_\tau, \mathbf{v} \rangle)$. The log moment-generating-function bound for a variable in $[-M, M]$ gives

$$\log\left(\frac{W_{t+1}}{W_t}\right) \leq \eta \langle \mathbf{g}_t, \mathbf{x}_t \rangle + \frac{\eta^2 M^2}{2}.$$

Sum over t , use $W_1 = |V|$, and lower bound W_{T+1} by the exponential weight of the best vertex. $\square$

L16.2.2 Main result

An explicit implementation of Algorithm L16.1 uses time and storage proportional to $|V|$. The main result replaces this dependence by evaluations of a function determined by the combinatorial structure of V .

Theorem L16.3 (Kernelization). There is a kernel $K_V : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that each round of vertex MWU or OMWU can be implemented using $d + 1$ evaluations of K_V , together with $O(d)$ scalar operations and $O(d)$ stored coordinates. The implementation produces exactly the same mean strategies under exact arithmetic. The kernel evaluator's own time and workspace, and the cost of obtaining the gradient, are additional.

We call the implementation *kernelized MWU* (KMWU) or *kernelized OMWU* (KOMWU). The theorem does not assert that every binary strategy set has an efficient kernel. Instead, it reduces efficient learning to a concrete computational question about V .

Corollary L16.4. If K_V can be evaluated efficiently, then vertex MWU/OMWU can be simulated efficiently even when $|V|$ is exponential in d .

The next three subsections prove Theorem L16.3: first define the kernel, then encode the entire weight vector implicitly, and finally recover its expectation.

L16.2.3 The 0/1-polyhedral kernel

Definition L16.5 (0/1-polyhedral feature map). For $\mathbf{z} \in \mathbb{R}^d$, define the vector $\varphi_V(\mathbf{z}) \in \mathbb{R}^V$ by

$$\varphi_V(\mathbf{z})_v = \prod_{k:v_k=1} z_k \quad (v \in V).$$

An empty product is one. Each coordinate of the feature map is a monomial corresponding to one binary strategy.

The feature map may have exponentially many coordinates. We introduce it to describe the computation, without constructing it explicitly.

Definition L16.6 (0/1-polyhedral kernel). The kernel is the inner product of two feature maps:

$$K_V(\mathbf{z}, \mathbf{w}) = \langle \varphi_V(\mathbf{z}), \varphi_V(\mathbf{w}) \rangle = \sum_{v \in V} \prod_{k:v_k=1} z_k w_k.$$

For the hypercube, for instance, this sum includes one monomial for each subset of coordinates. Its apparent exponential size will disappear when we factor it in the final section.

L16.2.4 Keeping track of the distribution over vertices

The update in Algorithm L16.1 adds a linear score to the exponent of each vertex's weight. Those scores can be stored in just d coordinates.

Theorem L16.7 (Implicit weights). Let $\mathbf{h}_t = \mathbf{g}_t$ for MWU and $\mathbf{h}_t = 2\mathbf{g}_t - \mathbf{g}_{t-1}$ for OMWU. Define, coordinatewise,

$$b_{t,k} = \exp\left(\eta \sum_{\tau=1}^{t-1} h_{\tau,k}\right).$$

At every round, the distribution maintained by vertex MWU/OMWU satisfies $\lambda_t(\mathbf{v}) \propto \varphi_V(\mathbf{b}_t)_v$.

Proof. We induct on t . Initially $\mathbf{b}_1 = \mathbf{1}$, so every coordinate of $\varphi_V(\mathbf{b}_1)$ equals one, giving the uniform distribution.

For the inductive step, the binary entries of $\mathbf{v}$ imply

$$\exp(\eta \langle \mathbf{h}_t, \mathbf{v} \rangle) = \prod_{k:v_k=1} \exp(\eta h_{t,k}).$$

Substituting the inductive hypothesis in the vertex update yields

$$\begin{aligned} \lambda_{t+1}(\mathbf{v}) &\propto \varphi_V(\mathbf{b}_t)_v \exp(\eta \langle \mathbf{h}_t, \mathbf{v} \rangle) \\ &= \prod_{k:v_k=1} (b_{t,k} \exp(\eta h_{t,k})) \\ &= \varphi_V(\mathbf{b}_{t+1})_v. \end{aligned}$$

This proves the claim for both choices of $\mathbf{h}_t$. □

In particular, we can update the implicit weights using only

$$b_{t+1,k} = b_{t,k} \exp(\eta h_{t,k}) \quad (k = 1, \dots, d).$$

There is no need to sum the entire gradient history at each round.

Corollary L16.8 (Normalization). The exact distribution over vertices is

$$\lambda_t(\mathbf{v}) = \frac{\varphi_V(\mathbf{b}_t)_v}{K_V(\mathbf{b}_t, \mathbf{1})}.$$

Proof. Since $\varphi_V(\mathbf{1})_v = 1$ for every vertex, the sum of the unnormalized weights is

$$\sum_{v \in V} \varphi_V(\mathbf{b}_t)_v = \langle \varphi_V(\mathbf{b}_t), \varphi_V(\mathbf{1}) \rangle = K_V(\mathbf{b}_t, \mathbf{1}).$$

This quantity is positive because V is nonempty and all coordinates of $\mathbf{b}_t$ are positive. □

L16.2.5 Reconstructing the expectation

Encoding the distribution is only half the task. `NextStrategy()` must also recover its mean without enumerating the vertices. The following identity supplies that step, extending the path-kernel idea of Takimoto, E., & Warmuth, M. K. [TW03].

Theorem L16.9 (Recovering the mean). For each k , let $\bar{e}_k = \mathbf{1} - e_k$, the vector with a zero in coordinate k and ones elsewhere. Then

$$x_{t,k} = \sum_{v \in V} \lambda_t(v) v_k = 1 - \frac{K_V(\mathbf{b}_t, \bar{e}_k)}{K_V(\mathbf{b}_t, \mathbf{1})}.$$

The same denominator is used for every coordinate, so $d + 1$ kernel evaluations recover the entire mean.

Proof. A monomial survives the zero in coordinate k exactly when its vertex has $v_k = 0$. Thus

$$\varphi_V(\bar{e}_k)_v = \begin{cases} 0 & \text{if } v_k = 1 \\ 1 & \text{if } v_k = 0 \end{cases} = 1 - v_k.$$

It follows that

$$\begin{aligned} K_V(\mathbf{b}_t, \bar{e}_k) &= \sum_{v: v_k=0} \varphi_V(\mathbf{b}_t)_v \\ &= K_V(\mathbf{b}_t, \mathbf{1}) \sum_{v: v_k=0} \lambda_t(v). \end{aligned}$$

Dividing by the normalizer gives the probability that coordinate k is zero. Its complement is the expected value of that binary coordinate. $\square$

Together, Theorem L16.7 and Theorem L16.9 prove Theorem L16.3. Maintain $\mathbf{b}_t$, evaluate one common normalizer and d coordinate-exclusion kernels, return the resulting mean, and update $\mathbf{b}_t$ after observing the gradient. No explicit vector indexed by V is needed.

L16.3 Examples of efficiently computable kernels

We now turn the abstract reduction into algorithms for the strategy sets introduced above. It is convenient to write $q_k = z_k w_k$ and

$$Z(\mathbf{q}) = \sum_{v \in V} \prod_{k: v_k=1} q_k, \quad K_V(\mathbf{z}, \mathbf{w}) = Z(\mathbf{q}).$$

The recurrences below evaluate the kernel for arbitrary real inputs. When $\mathbf{q} = \mathbf{b}_t > 0$, $Z(\mathbf{q})$ is also the normalizing partition function of the learning distribution.

L16.3.1 Hypercube

For $V = \{0, 1\}^d$, each coordinate is either absent from a monomial or contributes q_k . Distributing the product gives

$$K_V(\mathbf{z}, \mathbf{w}) = \prod_{k=1}^d (1 + z_k w_k).$$

The kernel takes $O(d)$ arithmetic operations to evaluate. For positive weights, the mean has the particularly simple form $x_k = \frac{b_k}{1+b_k}$.

L16.3.2 Multiple choices: m-sets

For subsets of size m , the kernel is the coefficient of γ^m in

$$\prod_{k=1}^d (1 + \gamma z_k w_k).$$

Each choice of m factors contributes exactly the monomial for that subset. Let $E_{j,r}$ be the coefficient of γ^r after the first j factors. Then

$$E_{j,r} = E_{j-1,r} + q_j E_{j-1,r-1}.$$

Initialize $E_{0,0} = 1$ and all invalid entries to zero. Only degrees through m are needed, so this dynamic program evaluates $K_V(\mathbf{z}, \mathbf{w}) = E_{d,m}$ in $O(d(m+1))$ operations. The recurrence separates subsets that omit object j from those that include it.

L16.3.3 Paths in a directed acyclic graph

For a directed acyclic graph $G = (N, E)$, let Z_u be the sum of products of edge weights over all paths from node u to the destination. In reverse topological order, compute

$$Z_{\text{destination}} = 1, \quad Z_u = \sum_{e=(u,v)} q_e Z_v \quad (u \neq \text{destination}).$$

The kernel is Z_{source} . The empty path at the destination contributes one; outgoing edges of the destination are ignored. A node with no route to the destination contributes zero.

Every source-to-destination path has a unique first edge, which proves the recurrence. Evaluating it takes $O(|N| + |E|)$ operations. This is the path-kernel setting studied by [Takimoto, E., & Warmuth, M. K. \[TW03\]](#) acyclicity is what makes this single backward pass sufficient.

L16.3.4 Resource allocations

For the one-hot allocation representation, let $F_j(r)$ be the total weight of allocations of r units to the first j battlefields. With $q_{j,s} = z_{j,s} w_{j,s}$, the recurrence is

$$F_0(0) = 1, \quad F_0(r) = 0 \quad (r > 0),$$

$$F_j(r) = \sum_{s=0}^r q_{j,s} F_{j-1}(r-s).$$

The last battlefield receives s units and contributes $q_{j,s}$; the preceding battlefields receive the remaining $r-s$ units. Consequently $K_V(\mathbf{z}, \mathbf{w}) = F_B(R)$. The computation uses $O(B(R+1)^2)$ operations, polynomial in the size of the one-hot representation.

L16.3.5 Extensive-form games

We finish with the sequence-form polytope. Besides evaluating one kernel in linear time, we will share computation across the $d + 1$ evaluations to implement an entire learning round in linear time.

Fix a player in a finite perfect-recall game. Let I range over this player's information sets, with action set $A(I)$. Coordinate Ia is the sequence ending in action a at I . Perfect recall gives I a unique preceding own sequence $p(I)$.

Let $C(I, a)$ contain the information sets whose preceding sequence is Ia , and let R contain the information sets whose preceding sequence is empty. Different members of $C(I, a)$ correspond to different observations; a pure realization plan specifies a continuation in *all* of them. The sequence-form constraints are

$$\sum_{a \in A(I)} x_{Ia} = x_{p(I)}, \quad x_{Ia} \geq 0, \quad x_{\emptyset} = 1.$$

We omit the fixed empty-sequence coordinate from the kernel. For a subtree rooted at I , let V_I be its reduced pure continuation plans *conditional on the preceding sequence being selected*. Each plan chooses one action at I , specifies continuations below that action, and puts zero on branches below unselected actions.

■ **One kernel evaluation.** Define the partial kernel

$$K_I(\mathbf{z}, \mathbf{w}) = \sum_{v \in V_I} \prod_{k: v_k=1} z_k w_k.$$

Conditional continuation plans are essential here: an unconditional projection of all full-game vertices could also contain the all-zero vector from plans that never reach I .

Theorem L16.10 (Sequence-form kernel recurrence). For any $\mathbf{z}, \mathbf{w} \in \mathbb{R}^d$, the partial kernels satisfy

$$K_I(\mathbf{z}, \mathbf{w}) = \sum_{a \in A(I)} \left(z_{Ia} w_{Ia} \prod_{J \in C(I, a)} K_J(\mathbf{z}, \mathbf{w}) \right),$$

and the full kernel is

$$K_V(\mathbf{z}, \mathbf{w}) = \prod_{I \in R} K_I(\mathbf{z}, \mathbf{w}).$$

These recurrences evaluate the kernel in linear time in the sequence-form representation.

Proof. A continuation plan at I selects exactly one action a . Conditional on that choice, the continuation plans at the information sets in $C(I, a)$ can be chosen independently. Their monomial weights multiply, along with the selected coordinate $z_{Ia} w_{Ia}$. Summing over actions gives K_I . The plans at distinct roots can also be chosen independently, yielding the product for K_V .

Evaluate information sets from descendants to ancestors. Each action and parent-child incidence is processed a constant number of times, so the total arithmetic cost is linear. $\square$

Remark L16.11. The child set depends on the chosen action: it is $C(I, a)$. Multiplying by every descendant of I regardless of a would count choices on branches that the player's own action has ruled out.

Using this linear-time evaluator independently for each of the $d + 1$ kernels already gives a polynomial-time implementation of vertex MWU/OMWU. It would, however, cost $O(d^2)$ per round. The special relationship among the required evaluations allows a faster implementation.

■ **A whole iteration in linear time.** Set $\mathbf{z} = \mathbf{b}_t$ and $\mathbf{w} = \mathbf{1}$. A backward pass computes and stores

$$B_{Ia} = b_{t, Ia} \prod_{J \in C(I, a)} Z_J, \quad Z_I = \sum_{a \in A(I)} B_{Ia}.$$

Here $Z_I = K_I(\mathbf{b}_t, \mathbf{1})$ and the total normalizer is $\prod_{I \in R} Z_I$. All these weights are positive. The conditional probability of selecting action a , given the preceding sequence, is $\frac{B_{Ia}}{Z_I}$.

A forward pass starts with $x_{t, \emptyset} = 1$ and visits information sets from ancestors to descendants, setting

$$x_{t, Ia} = x_{t, p(I)} \frac{B_{Ia}}{Z_I}.$$

Every child information set in $C(I, a)$ inherits the same preceding-sequence realization weight $x_{t, Ia}$. We do not divide that weight among observations. The product decomposition used in Theorem L16.10 proves that these are exactly the coordinate marginals of λ_t . In particular, $\sum_{a \in A(I)} x_{t, Ia} = x_{t, p(I)}$.

The backward and forward passes each take linear time. After observing the gradient, update the d coordinates of $\mathbf{b}_t$ as before. This computes all the expectations required by Algorithm L16.1 without repeating a separate traversal for each coordinate.

Example L16.12 (A branch with a continuation). At I , action a ends the player's decisions, while action b leads to an information set J with actions c, d . In coordinates (a, b, c, d) , the vertices are

$$(1, 0, 0, 0), \quad (0, 1, 1, 0), \quad (0, 1, 0, 1).$$

For weights $(2, 3, 5, 7)$, the backward pass gives $Z_J = 5 + 7 = 12$ and $Z_I = 2 + 3(5 + 7) = 38$. The forward pass gives

$$\mathbf{x} = \left(\frac{2}{38}, \frac{36}{38}, \frac{15}{38}, \frac{21}{38} \right).$$

In particular, $x_c + x_d = x_b$. Multiplying the continuation factor into both root actions would instead give the incorrect normalizer $(2 + 3)(5 + 7) = 60$.

Corollary L16.13 (Linear-time vertex learning in extensive-form games). For each player, vertex MWU/OMWU can be implemented exactly with arithmetic cost linear in that player's sequence-form representation per round, beyond the cost of computing the payoff gradient. Its regret and equilibrium-convergence guarantees transfer under the corresponding assumptions of the vertex algorithm.

This recovers learning over the reduced normal-form strategies without explicitly constructing that normal-form game. It is a different update from CFR, which combines local regret minimizers. The advantage here is that results about the vertex learning dynamics can be used directly.

The exact identities above use real arithmetic. In an implementation, logarithms and log-sum-exp prevent overflow in the positive weights; a finite-precision guarantee must also account for rounding error.

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, pp. 6337–6357. [Online]. Available: <https://proceedings.mlr.press/v162/farina22a.html>
- [TW03] E. Takimoto and M. K. Warmuth, “Path kernels and multiplicative updates,” *Journal of Machine Learning Research*, vol. 4, pp. 773–818, 2003.