6.7980: Topics in Multiagent Learning

Lecture 13 – Pushing the boundary

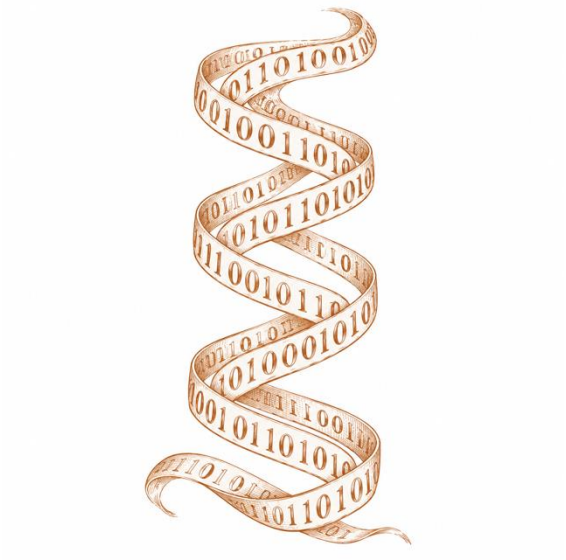
Prof. Daskalakis

Fall 2026



Combinatorial Games





vs





Games where the actions of each player i are bit-strings $A_i \subseteq \{0,1\}^{d_i}$ defining some combinatorial object and there could be exponentially many of those..

Combinatorial Game Examples

Normal-Form Games

- Deterministic strategies of each player are in one-to-one correspondence with vertices of the simplex in $|A_i|$ dimensions

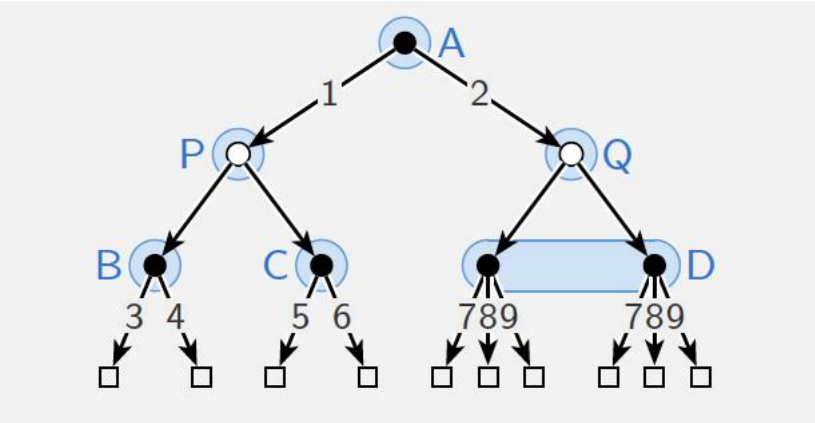
$$\begin{pmatrix}1\\0\\0\\\vdots\\0\end{pmatrix}, \quad \begin{pmatrix}0\\1\\0\\\vdots\\0\end{pmatrix}, \quad \begin{pmatrix}0\\0\\1\\\vdots\\0\end{pmatrix}, \quad \ldots, \quad \begin{pmatrix}0\\0\\0\\\vdots\\1\end{pmatrix}.$$

- Number of bit strings = number of actions

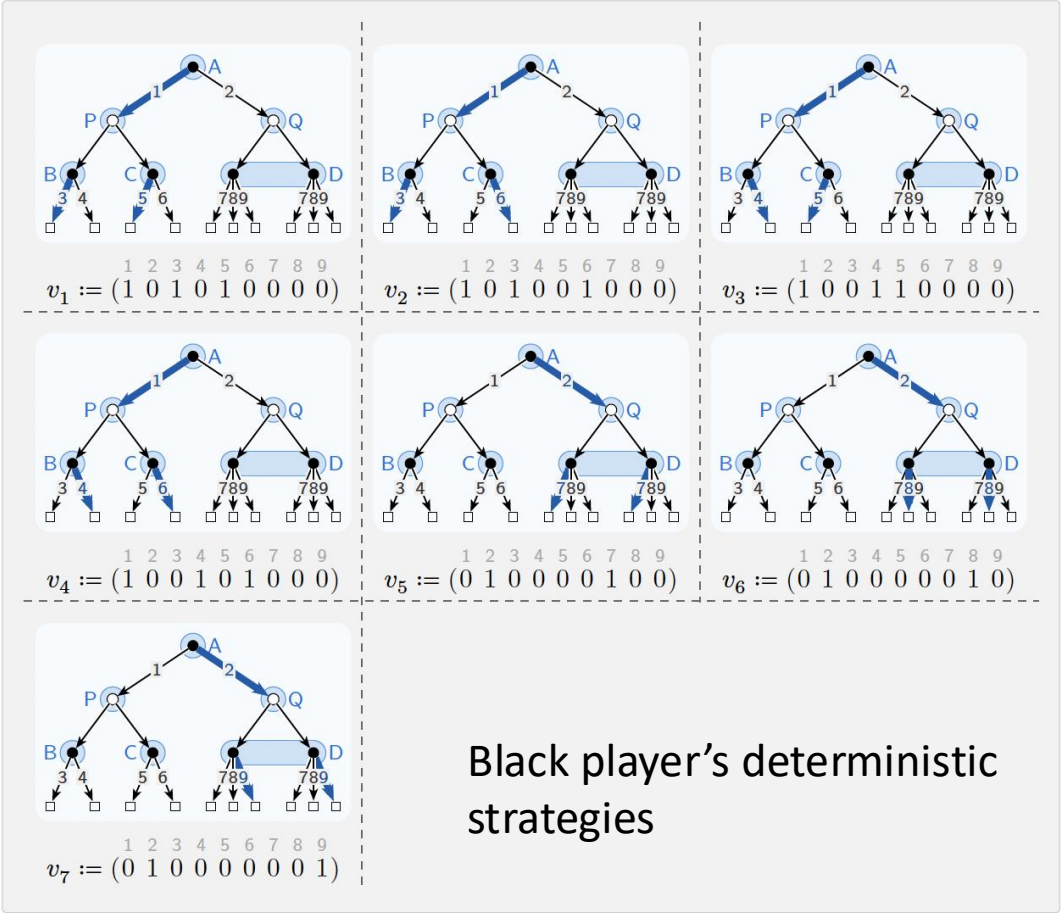
Combinatorial Game Examples

Extensive-Form Games

- Deterministic sequence-form strategies are bit strings corresponding to 0/1 probability flows in each player's tree-form decision process



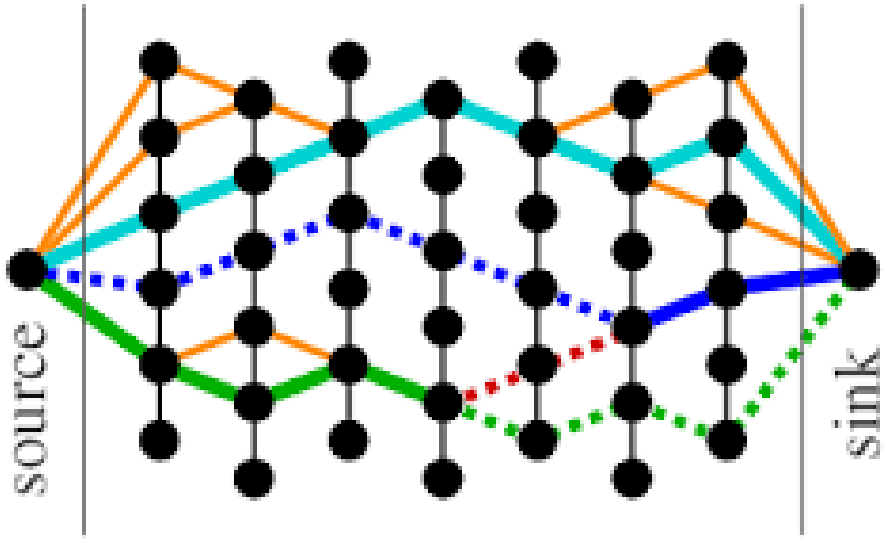
- Number of strategies could be exponential in game representation



Combinatorial Game Examples

Path-Games

- Deterministic strategies are all simple paths from source to sink
- Can be represented as a bit string of which edges are used (1) or not used (0) in the path

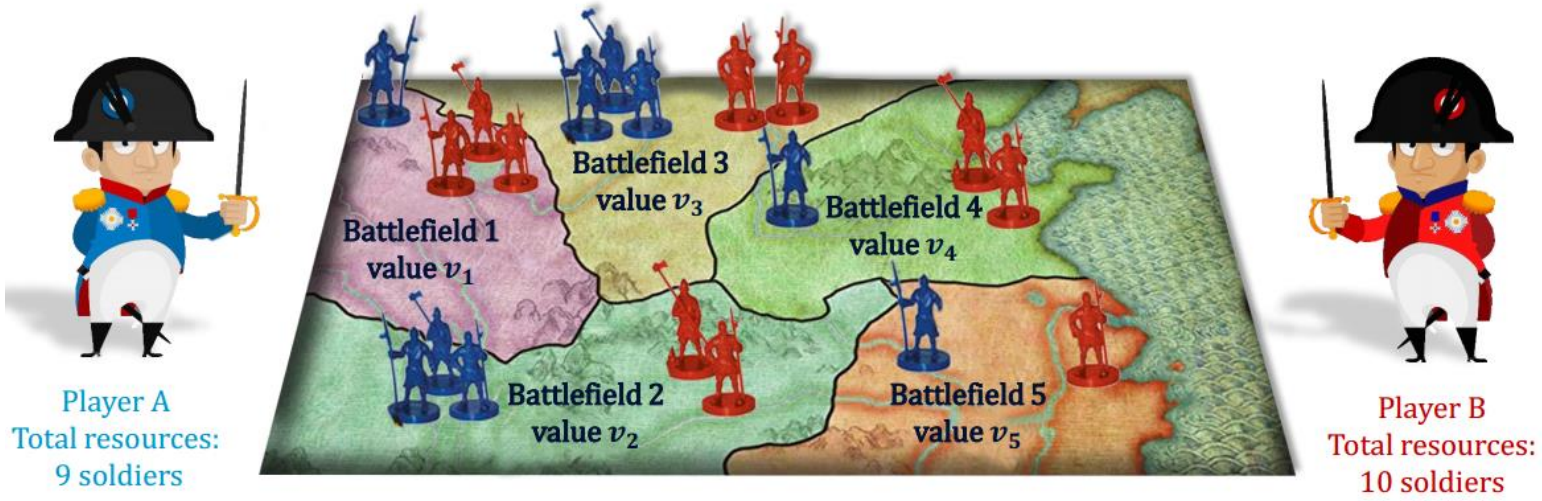


- Number of strategies per player could be exponentially many in the graph size

Combinatorial Game Examples

Colonel Blotto

- Deterministic strategies are all ways to split n soldiers into k battlefields
- Can be represented as a bit string
  - e.g. one block per battlefield, and for each block a one-hot encoding of number of soldiers allocated



- Number of strategies per player: if n soldiers, k battlefields: $\binom{n+k-1}{k-1}$

General Setting

Each player chooses strategy $x_i \in \Omega_i \subseteq \mathbb{R}^{d_i}$ where Ω_i is polyhedral strategy set with 0/1 vertices V_i

Utility of player i is linear in x_i's

Goal: no-regret learning

Challenge: How to sidestep exponential size of V_i?

- If we can project into Ω_i can use projected gradient descent
- But this could be far from optimal, e.g. even in normal-form game $\sqrt{d_i T}$ vs $\sqrt{\log d_i T}$

No-Regret Algorithms for Combinatorial Games

Different conceptual approaches exist:

Exploits structure of problem and specific learning algorithm

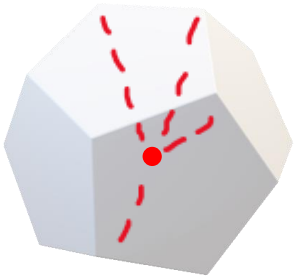
Conversion to a single simplex of convex combinations of vertices

Decomposition into local decision problem over actions at each decision point

Use general convex optimization tools (e.g., FTRL)

Less specialized; general tool

Main idea:



Key question:

How to sidestep exponential size?

Every point in the polytope is a convex combination of its finitely many vertices $V := \{v_1, \dots, v_m\}$. So, operate a change of **variable:** learn the convex combination, not the points $x^{(t)}$

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

Perf. of vertex

$$R^{(T)} := \max_{\hat{\lambda} \in \Delta(V)} \sum_{t=1}^{T} \left\langle \begin{pmatrix} \vdots \\ \langle u^{(t)}, v \rangle \\ \vdots \end{pmatrix}, \hat{\lambda} - \lambda^{(t)} \right\rangle$$

General Setup:

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

V_i vertices of Ω_i

Vertex MWU algorithm

Setup
$\Omega_i \subseteq \mathbb{R}^d$
V_i vertices of Ω_i

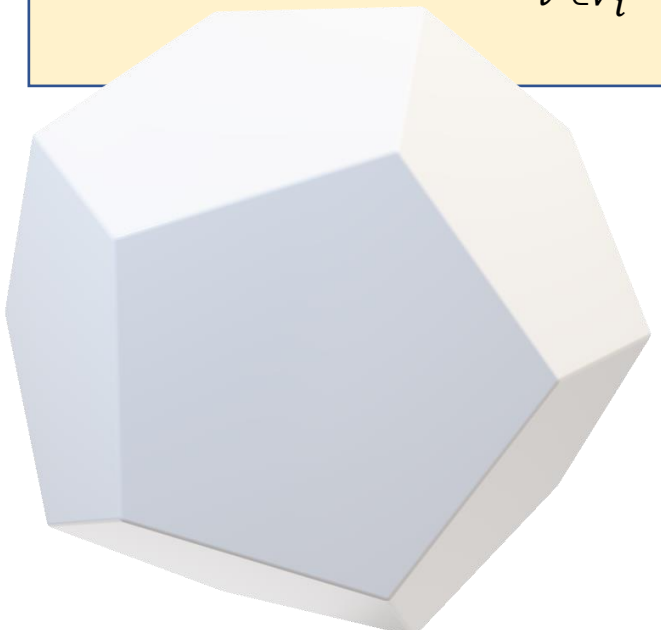
$$\lambda^{(1)} := \frac{1}{|V_i|}\mathbf{1} \in \mathbb{R}^{V_i}$$

For $t = 1, 2, \ldots$

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

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

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



"Utility of vertex v"

…We weight vertices using MWU

Main theorem [Farina-Lee-Luo-Kroer'22]

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

Vertex MWU algorithm

Setup
$\Omega_i \subseteq \mathbb{R}^d$
V_i vertices of Ω_i

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

For $t = 1, 2, \ldots$

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

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

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

Crucially independent on the number of vertices of Ω_i!

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

Setup

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

V vertices of Ω

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

Definition (0/1-feature map of Ω)

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

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

Definition (0/1-polyhedral kernel of Ω)

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

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

Idea #1

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

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

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

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

Proof: by induction

Vertex MWU algorithm

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

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

For $t = 1, 2, \ldots$

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

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

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

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

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

Idea #1

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

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

$$\left(\sum_{\tau=1}^{t-1} \right)$$

Pr

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

Vertex MWU algorithm

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

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

For $t = 1, 2, \ldots$

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

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

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

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

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

Idea #2

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

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

Vertex MWU algorithm

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

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

For $t = 1, 2, \ldots$

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

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

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

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

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

(d+1 kernel evaluations)

Vertex MWU algorithm

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

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

For $t = 1, 2, \ldots$

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

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

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

Kernelized MWU algorithm

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

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

For $t = 1, 2, \ldots$

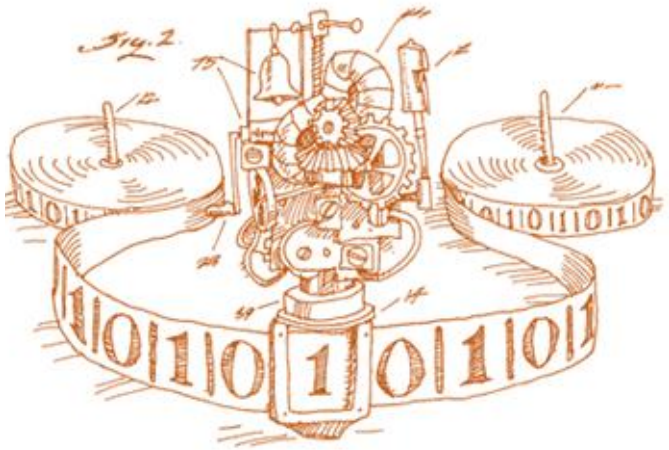
Play $x^{(t)} \coloneqq \left(1 - \frac{K_\Omega(b^{(t)}, \mathbf{1}-e_1)}{K_\Omega(b^{(t)}, \mathbf{1})}, \ldots, 1 - \frac{K_\Omega(b^{(t)}, \mathbf{1}-e_d)}{K_\Omega(b^{(t)}, \mathbf{1})}\right)$

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

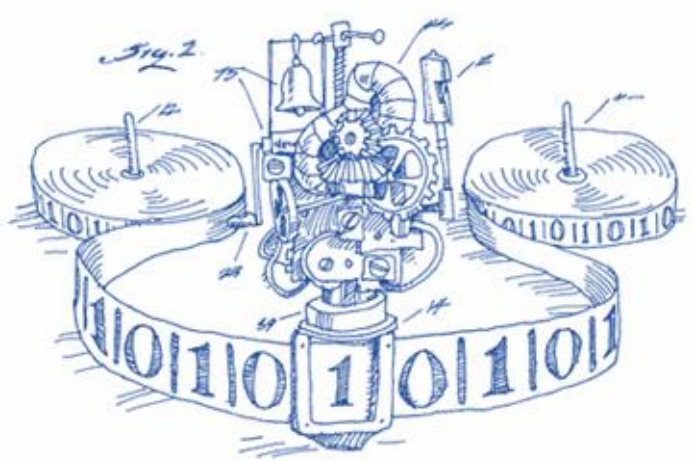
Set $b^{(t+1)} \coloneqq \exp\left\{\eta \sum_{\tau=1}^{t} u^{(\tau)}\right\}$

Non-Compact Games





vs





What if players can write code they want, innovate on morphology of their robot/molecule etc? The corresponding strategy spaces become non-compact or infinite-dimensional/non-parametric.

Equilibrium existence and standard equilibrium learning dynamics fail, in general.

Q: How to resurrect their existence?

Why does equilibrium existence fail?





Player 2 (max player)

Player 1 (min player)

	1	2	3	4	…
1	1	1	1	1	…
2	-1	1	1	1	…
3	-1	-1	1	1	…
4	-1	-1	-1	1	…
…	…	…	…	…	…

"Guess the larger number" Game

Fact: "Guess the larger number" game has no Nash equilibrium (not even a non-trivial approximate eq).

- what if we **exclude** "guess the larger number"?

Resurrecting Equilibrium Existence



Player 2 (max player)



Player 1 (min player)

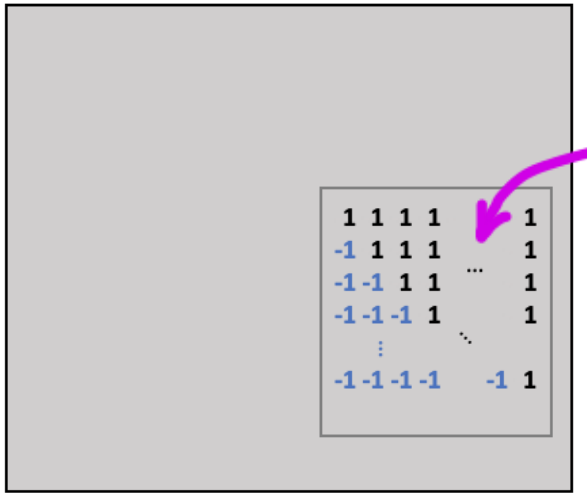
	1	2	3	4	…
1	1	1	1	1	…
2	-1	1	1	1	…
3	-1	-1	1	1	…
4	-1	-1	-1	1	…
…	…	…	…	…	…



Player 2 (max player)



Player 1 (min player)



Fact: "Guess the larger number" game has no Nash equilibrium (not even a non-trivial approximate eq).

Theorem [Hanneke-Livni-Moran'21]: If an (infinite) {−1,1}-valued two-player zero-sum game has no subgame which is "Guess the larger number," then it has an ϵ-approximate Nash equilibrium for all $\epsilon > 0$.

- how about **real-valued** games?

Resurrecting Equilibrium Existence

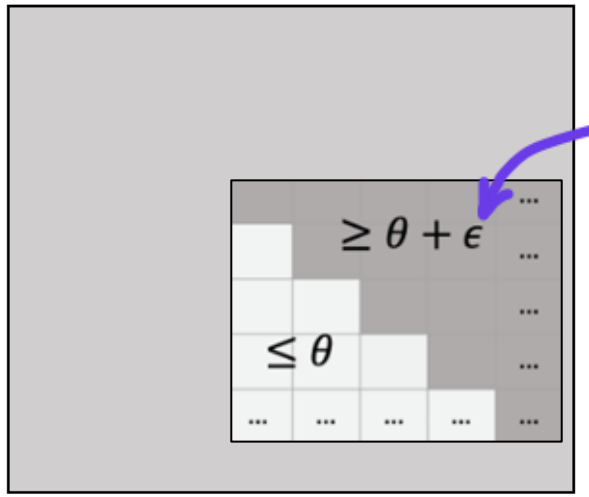
Player 2 (max player)

Player 1 (min player)

	1	2	3	4	…
1	1	1	1	1	…
2	-1	1	1	1	…
3	-1	-1	1	1	…
4	-1	-1	-1	1	…
…	…	…	…	…	…

Player 2 (max player)

Player 1 (min player)



Fact: "Guess the larger number" game has no Nash equilibrium (not even a non-trivial approximate eq).

Theorem [Hanneke-Livni-Moran'21]: If an (infinite) $\{-1,1\}$-valued two-player zero-sum game has no subgame which is "Guess the larger number," then it has an ϵ-approximate Nash equilibrium for all $\epsilon > 0$.

Theorem [Daskalakis-Golowich'22] For any $\epsilon > 0$, if an (infinite) real-valued two-player zero-sum game has no subgame which is ϵ-close to some "scaling" of "Guess the larger number," then it has $C \cdot \epsilon$-approximate Nash equilibrium.

Curious fact: existence proofs use **Ramsey's theorem** rather than **Brouwer's fixed point theorem** **!?!**

Learning Dynamics

Double Oracle Algorithm [McMahan-Gordon-Blum'03] (also used in PSRO)

Double Oracle Algorithm

Setting: two-player zero-sum game $G = (\mathcal{A}, \mathcal{B}, u)$, $\mathcal{A}$: minimizer's strategies
Input: nonempty finite subsets $A_0 \subseteq \mathcal{A}$, $B_0 \subseteq \mathcal{B}$, and $\varepsilon \geq 0$

1: Let t: $= 0$
2: **repeat**
3: Find a minimax equilibrium (p_t^*, q_t^*) of subgame (A_t, B_t, u)
4: Find some $a_{t+1} \in \mathrm{BR}_{\mathcal{A}}(q_t^*)$ and $b_{t+1} \in \mathrm{BR}_{\mathcal{B}}(p_t^*)$
5: Let $A_{t+1} := A_t \cup \{a_{t+1}\}$ and $B_{t+1} := B_i \cup \{b_{t+1}\}$
6: $t := t + 1$
7: **end if** $u(p_t^*, b_{t+1}) - u(a_{t+1}, q_t^*) \leq \varepsilon$

Output: ε-equilibrium (p_t^*, q_t^*) of game G

Question (also asked in **[Gemp et al.'22]**)**:** under what conditions does this end in finite time?
How about multi-player/general-sum generalizations of this algorithm?

Computing equilibria

- **Setting:** an infinite zero-sum game (+ extensions to general-sum in our paper)
- **Goal:** compute a minimax equilibrium using an easy-to-compute oracle
- We'll assume access to two oracles:
  - **Value oracle:** given strategies for both players, output the utility (verifier solves this!)
  - **Best-response (aka ERM) oracle:** given a finitely-supported mixed strategy of the opponent, returns a best response (Claude solves this given verifier!)

Theorem [Assos, Attias, Dagan, Daskalakis, Fishelson '22]: There is a (variation to the double-oracle) algorithm that computes an ϵ-minimax equilibrium using a best-response and a value oracle for both players, as long as the guess-the-larger-number obstruction is absent from the game (in which case there do not exist equilibria).

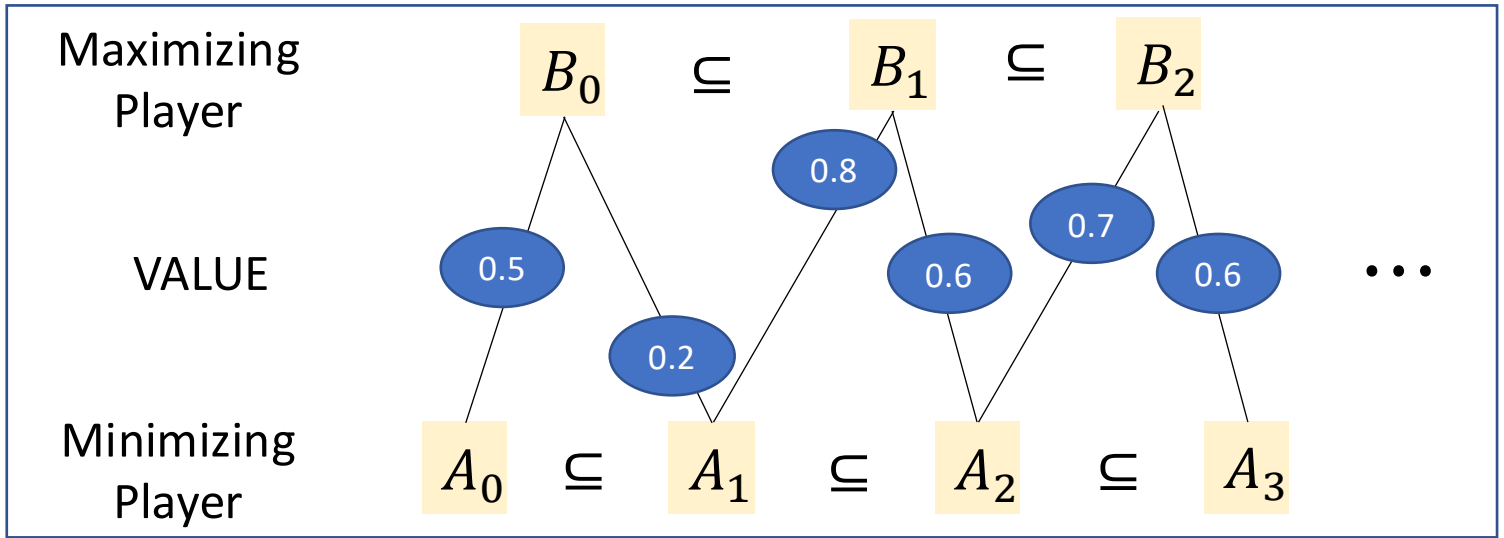
Follows from [Hazan, Koren '16]: The running time of our algorithm is optimal if the only access to the game is a value and a best-response oracle.

Algorithm: a variant of Double-Oracle

- A "Turn-based" Double Oracle algorithm
- The algorithm computes action sets $A_0 \subseteq A_1 \subseteq \cdots \subseteq \mathcal{A}$ for the minimizing player (player w/ action set $\mathcal{A}$) and $B_0 \subseteq B_1 \subseteq \cdots \subseteq \mathcal{B}$ for the maximizing player (player w/ action set $\mathcal{B}$) such that

$$\mathrm{Val}(A_{t+1}, B_t) = \mathrm{Val}(\mathcal{A}, B_t) \leq \mathrm{Val}(A_t, B_t) - \epsilon$$
$$\mathrm{Val}(A_{t+1}, B_{t+1}) = \mathrm{Val}(A_{t+1}, \mathcal{B}) \geq \mathrm{Val}(A_{t+1}, B_t) + \epsilon$$

- Each iteration is implemented using **Best-Response** and **Value** oracle calls
- **Central Claim:** The algorithm is guaranteed to terminate after $2^{O\left(tr(G)^3/\epsilon^4\right)}$ iterations where $tr(G)$ is largest guess-the-largest game!
  - An ϵ-minimax equilibrium can be computed from there!



Computing B_{t+1} (similarly A_{t+1})

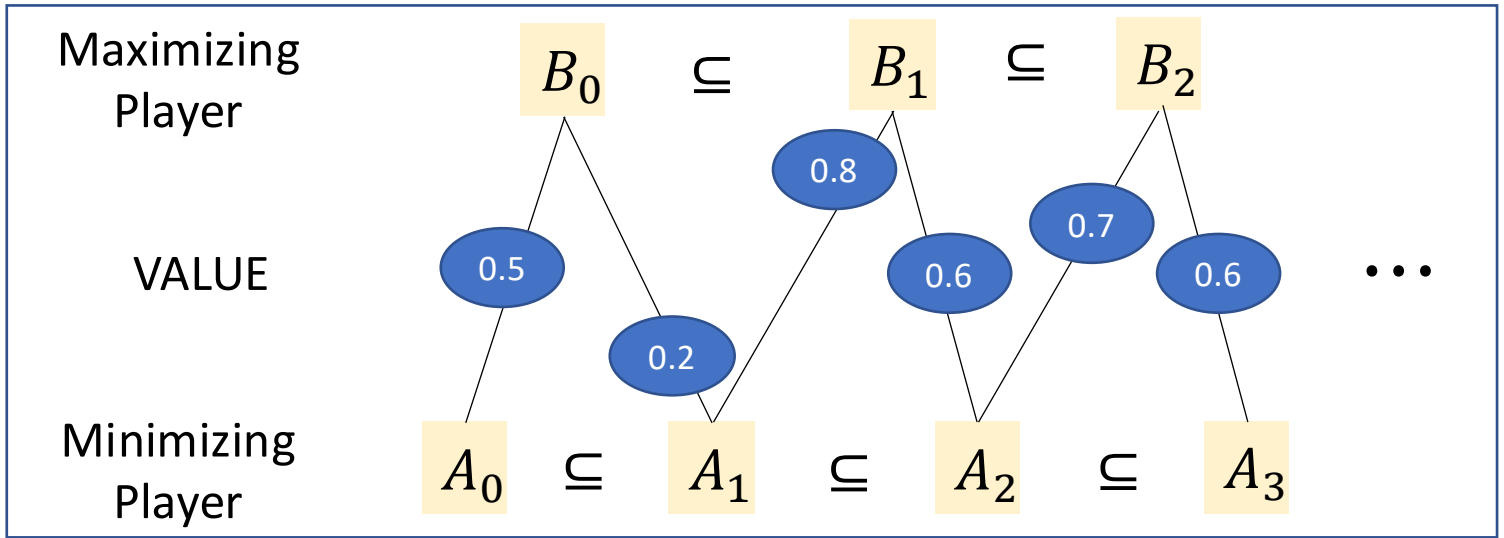
- Alternatingly, over multiple rounds, Player A updates her randomization over A_{t+1} (which is finite!) using a no-regret learning algorithm, and Player B plays her best-response over the full set $\mathcal{B}$ (using ERM oracle!) against A's average history so far (i.e. runs Be-The-Leader algorithm)
- $B_{t+1} \leftarrow B_t \cup \{\text{actions played by Player B}\}$

Algorithm: a variant of Double-Oracle

- A "Turn-based" Double Oracle algorithm
- The algorithm computes action sets $A_0 \subseteq A_1 \subseteq \cdots \subseteq \mathcal{A}$ for the minimizing player (player w/ action set $\mathcal{A}$) and $B_0 \subseteq B_1 \subseteq \cdots \subseteq \mathcal{B}$ for the maximizing player (player w/ action set $\mathcal{B}$) such that

$$\mathrm{Val}(A_{t+1}, B_t) \approx \mathrm{Val}(\mathcal{A}, B_t) \leq \mathrm{Val}(A_t, B_t) - \epsilon$$
$$\mathrm{Val}(A_{t+1}, B_{t+1}) \approx \mathrm{Val}(A_{t+1}, \mathcal{B}) \geq \mathrm{Val}(A_{t+1}, B_t) + \epsilon$$

- Each iteration is implemented using **Best-Response** and **Value** oracle calls
- **Central Claim:** The algorithm is guaranteed to terminate after $2^{O\left(tr(G)^3/\epsilon^4\right)}$ iterations where $tr(G)$ is largest guess-the-largest game!
  - An ϵ-minimax equilibrium can be computed from there!



Computing B_{t+1} (similarly A_{t+1})

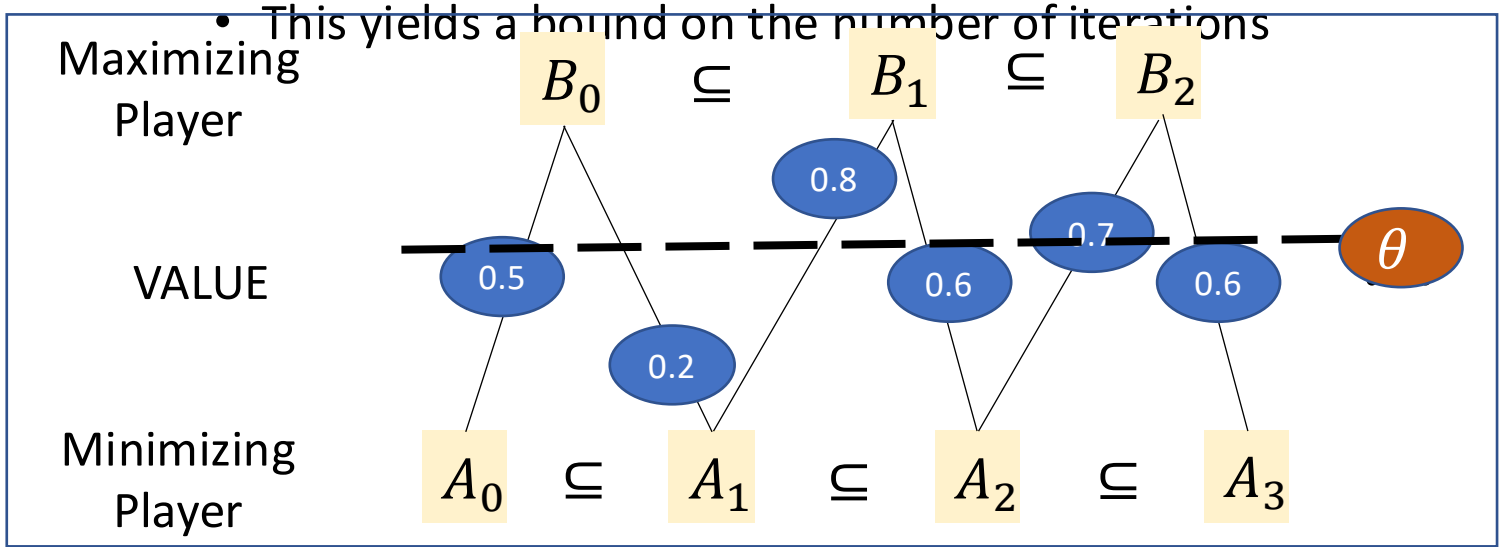
- Alternatingly, over multiple rounds, Player A updates her randomization over A_{t+1} (which is finite!) using a no-regret learning algorithm, and Player B plays her best-response over the full set $\mathcal{B}$ (using ERM oracle!) against A's average history so far (i.e. runs Be-The-Leader algorithm)
- $B_{t+1} \leftarrow B_t \cup \{\text{actions played by Player B}\}$

Analyzing the game: binary case

- Assume that the algorithm proceeds for T iterations (want to show must be finite)
- **Claim:** can find $t_1, t_2, \dots, t_{k=\Theta(T\epsilon)}$ and a threshold θ such that

$$\mathrm{Val}\left(A_{t_i}, B_{t_j}\right) \le \theta \text{ if } i > j \,;\, \mathrm{Val}\left(A_{t_i}, B_{t_j}\right) \ge \theta + \epsilon \text{ if } i \le j$$

- Hence, there exists an ϵ-separated "guess-the-larger-number" subgame of **mixed** strategies $p_{t_1}, \dots, p_{t_k}$, $q_{t_1}, \dots, q_{t_k}$, where (p_{t_i}, q_{t_i}) is minmax strategy of the finite subgame
  - By **[Hanneke-Livni-Moran '21]**, **[Assos, Attias, Dagan, Daskalakis, Fishelson '23]**, there exists a guess-the-larger-number subgame of **pure** strategies of size about $\log k$.* (*if time permits)
  - Since threshold dimension is bounded (a.k.a. Littlestone is bounded), the size of this subgame is bounded.
  - This yields a bound on the number of iterations



Player B (max player)

Player A (min player)

	q_{t_1}	q_{t_2}	q_{t_3}	q_{t_4}	…
p_{t_1}					…
p_{t_2}		$\ge \theta + \epsilon$			…
p_{t_3}					…
p_{t_4}	$\le \theta$				…
…	…	…	…	…	…

Analyzing the game: binary case

Large

	q_1	q_2	q_3	…	q_N
p_1					
p_2		$\geq \theta + \epsilon$			
p_3					
…	$\leq \theta$				
p_N					

$\Longrightarrow$

Large-ish

	b_1	b_2	b_3	…	b_n
a_1					…
a_2			1		…
a_3					…
…		-1			…
a_m	…	…	…	…	…

Here is where Ramsey is involved, $m \approx \log N$

General Results

Setting	Time per iter.	BR calls/it.	#iterations
Minmax, 0-1 valued	t/ϵ^4	$\log t/\epsilon^2$	$C^{\mathrm{Lit}(G)/\epsilon^2} \wedge \epsilon^{-C\,\mathrm{VC}(G)^2\,\mathrm{tr}(G)/\epsilon^4}$
Minmax, real valued	t/ϵ^4	$\log t/\epsilon^2$	$C^{\mathrm{sfat}(G,\epsilon)/\epsilon^2} \wedge \epsilon^{-CI(G)^2\,\mathrm{fattr}(G,\epsilon)/\epsilon^5}$
CCE, 0-1 valued	kt/ϵ^2	$k\log t/\epsilon^2$	$C^{(k/\epsilon^3)\,\mathrm{Lit}(G)} \wedge \epsilon^{-C(k^3/\epsilon^6)\,\mathrm{VC}(G)^2\,\mathrm{tr}(G)}$
CCE, real valued	kt/ϵ^2	$k\log t/\epsilon^2$	$C^{(k/\epsilon^3)\,\mathrm{sfat}(G,\epsilon)} \wedge \epsilon^{-C(k^3/\epsilon^6)I(G)^2\,\mathrm{fattr}(G,\epsilon)}$

Table 2: The table describes the time per iteration, the number of best-response calls per iteration and the number of iterations of our algorithms, up to polylogarithmic factors for finding an $O(\epsilon)$-approximate Nash in a zero-sum two player game (minmax equilibrium) and Coarse Correlated Equilibrium (CCE) in general games G. Here, $C > 0$ is a universal constant, and $\mathrm{Lit}, \mathrm{VC}, \mathrm{tr}, \mathrm{sfat}, \mathrm{fat}, \mathrm{fattr}$ denote Littlestone, VC, threshold, sequential fat, fat and fat-threshold dimensions of G, $I(G) = \int_0^1 \left(\sqrt{\mathrm{fat}(G,\delta)d\delta}\right)^2$ and $\wedge$ denotes a minimum of two terms.