15-888 Computational Game Solving (Fall 2025) Wednesday, September 10th, 2025

Lecture 5
Extensive-Form Games and Counterfactual Regret Minimization

Ioannis Anagnostides

Our last lecture covered some basic equilibrium computation algorithms in games represented in normal form. As alluded to, while every finite game can be cast into normal form, that representation is often inefficient. This is particularly so when one is modeling sequential interactions, as we shall see. A more compact representation in such settings is the *extensive form*. This lecture centers on extensive-form games and decision making therein, culminating in the *counterfactual regret minimization (*CFR*)* algorithm, which is at the heart of many practical breakthrough results.

In more detail, we will cover the following topics. To begin with, in Section 1, we will introduce the formalism of (imperfect-information) extensive-form games. We will focus especially on the question of how to represent strategies. The upshot is that, under a suitable, compact way of representing strategies, one can still express the utility of each player as a *linear function* with respect to its own strategy—when retaining the rest of the players' strategies fixed. This will enable us to import much of the technology developed for normal-form games. Section 2 introduces the framework of *tree-form decision problems (TFDPs)*, which results from an extensive-form game when one analyzes solely the decision problem faced by each player individually. We will see how the strategy set of each player can be decomposed inductively through a series of basic operations. Together with the framework of *regret circuits* (Section 3), we will see how one can minimize regret in a tree-form decision problem by relying solely on regret minimizers operating over a simplex. The resulting algorithm is CFR.

1 Extensive-form games

An *extensive-form game* is represented through a rooted *game tree*. The set of nodes in the tree is denoted by $\mathcal{H}$. $\mathcal{H}$ contains either *leaf* (that is, *terminal*) nodes or *decision* nodes. Every decision node is associated with a player that acts at that node. The set of all decision nodes that belong to player i is denoted by $\mathcal{H}_i \subseteq \mathcal{H}$. By convention, the set of players $[n] \cup \{c\}$ includes a (non-strategic) *chance player* who selects actions according to some fixed probability distribution; the chance player intends to model external stochasticity, such as the roll of a dice. The player who acts at a node $h \in \mathcal{H}$ is to select one of the available actions at that node, denoted by $\mathcal{A}_h$. The game tree contains an edge for each possible action $a \in \mathcal{A}_h$ that connects h to the node the game transitions to when a is acted upon. When the game transitions to a leaf node $z \in \mathcal{Z}$, each player $i \in [n]$ obtains a payoff specified by a utility function $u_i : \mathcal{Z} \to \mathbb{R}$.

Imperfect information and perfect recall To capture imperfect information, the nodes $\mathcal{H}_i$ of each player $i \in [n]$ are partitioned into *information sets* $\mathcal{J}_i$. Every information set $j \in \mathcal{J}_i$ contains all nodes that are indistinguishable for that player. We can thus assume that $\mathcal{A}_h = \mathcal{A}_{h'} = \mathcal{A}_j$

for all nodes h, h' that belong to the same information set $j \in \mathcal{J}_i$, for otherwise that player could distinguish between h and h'. The set of *sequences* Σ_i is defined as $\Sigma_i \coloneqq \{(j, a) : j \in \mathcal{J}_i, a \in \mathcal{A}_j\}$. For a node $h \in \mathcal{J}_i$, we denote by $\sigma_i(h)$ the ordered list of previous information sets encountered by that player and actions played at those information sets.

We will mostly assume that the game has *perfect recall*, which intuitively means that players never forget previously acquired information. Figure 1 gives an example of an imperfect-recall game. Player 1 has two information sets, I_1 and I_2. There are two available actions at I_1 (labeled R_1 and L_1) and two available actions at I_2 (labeled R_2 and L_2). By the time Player 1 gets to act at I_2, it has forgotten its action at I_1. Formally, we say that a player $i \in [n]$ has perfect if $\sigma_i(h) = \sigma_i(h')$ for all nodes h, h' that belong to the same information set $j \in \mathcal{J}_i$. A game is of perfect recall of all players therein have perfect recall.

1.1 Representation of strategies

A *pure strategy* for a player $i \in [n]$ is any function of the form $\mathcal{J}_i \ni j \mapsto \mathcal{A}_j$. We let Π_i be the set of deterministic strategies for player $i \in [n]$. $\Delta(\Pi_i)$ is the set of *mixed* strategies; in what follows, a mixed strategy will be denoted by $\boldsymbol{\mu}_i \in \Delta(\Pi_i)$.

Every extensive-form game can be cast into normal form as follows. We can construct a normal-form game—with the same set of players excluding the chance player—in which the pure strategies of a player i is Π_i and the utilities are in accordance with the original extensive-form game; namely, under a joint pure strategy $(\pi_1, \dots, \pi_n) \in \Pi_1 \times \cdots \times \Pi_n$,

$$\sum_{z \in \mathcal{Z}} u_i(z) p_c(z) \prod_{i' \in [n]} \prod_{\substack{(h,a) \preceq z \\ h \in \mathcal{H}_{i'}}} \pi_{i'}(a \mid h). \tag{1}$$

A few clarifications are in order.

- the notation $(h, a) \preceq z$ means that the path from the root to the leaf z contains (h, a);
- if $j \in \mathcal{J}_i$ is the information set for which $h \in j$, $\pi_i(a \mid h)$ is to interpreted as $\pi_i(a \mid j)$; and
- $p_c(z)$ is the probability that the chance player plays all actions on the path from root to z.

The key issue here is that $|\Pi_i| = \prod_{j \in \mathcal{J}_i} |\mathcal{A}_j|$, so the number of pure strategies grows exponentially with the number of information sets. As a concrete example, let's say we have two players. Player 1 acts first and selects an action from the set $[m]$. Player 2 observes the action of Player 1, and gets to play either L or R in either of its information sets. Even though the extensive-form game has $\Theta(m)$ nodes, the number of actions for Player 2 in the induced normal-form game is 2^m, which is clearly prohibitive. We summarize this issue below.

Observation 1.1. *There is a two-player extensive-form game with $\Theta(m)$ nodes whose induced normal-form representation requires at least 2^m actions for one of the players.*

This is not just a theoretical pathology: this type of example is ubiquitous whenever a player needs to account for a large number of possible contingencies. So we certainly cannot afford to

treat every extensive-form game in normal form; we will need to develop specialized techniques to tackle extensive-form games.

In light of the foregoing example, operating over the entire set of mixed strategies seems like a daunting task. Instead, we will seek alternative, more compact representations. From an optimization standpoint, there are two basic desiderata one should keep in mind.

1. The set of strategies needs to be a convex polytope described with a polynomial—in the size of the game tree—number of constraints.

2. each player's utility needs to be linear—or at least concave—in that player's strategy.

At the very least, when the rest of the players are fixed, one should be able to optimize one's utility efficiently.

Behavioral strategies and Kuhn's theorem A *behavioral strategy* $\boldsymbol{b}_i \in \bigtimes_{j \in \mathcal{J}_i} \Delta(\mathcal{A}_j)$ for a player $i \in [n]$ assigns at each action $a \in \mathcal{A}_j$ at information set j a probability. A behavioral strategy is a mixed strategy in which one mixes *independently at each information set.* The famous theorem of Kuhn [1953] implies that, for *perfect-recall* games, behavioral strategies are as expressive as mixed strategies in the following formal sense.

Theorem 1.2 (Kuhn, 1953)**.** *Consider any perfect-recall game. For any mixed strategy $\boldsymbol{\mu}_i \in \Delta(\Pi_i)$, there exists a behavioral strategy $\boldsymbol{b}_i \in \bigtimes_{j \in \mathcal{J}_i} \Delta(\mathcal{A}_j)$ such that $u_i(\boldsymbol{b}_i, \boldsymbol{b}_{-i}) = u_i(\boldsymbol{\mu}_i, \boldsymbol{b}_{-i})$ for all $\boldsymbol{b}_{-i}$.*

An example with imperfect recall The conclusion of Theorem 1.2 does not hold beyond games of perfect recall. Let's examine again the imperfect-recall game of Figure 1. We recall (no pun intended) that Player 1 has two information sets, I_1 and I_2. It has two actions at I_1 (R_1 and L_1) and two actions at I_2 (R_2 and L_2). The values at the terminal nodes represent the utility of Player 1. The game is zero-sum, so Player 2 is striving to minimize the utility of Player 1.

Now, if Player 1 is allowed to employ mixed strategies, Player 1 could choose $\boldsymbol{\mu}_1 \coloneqq \frac{1}{2}(\mathsf{L}_1, \mathsf{L}_2) + \frac{1}{2}(\mathsf{R}_1, \mathsf{R}_2)$; this involves correlating the randomization between I_1 and I_2, so $\boldsymbol{\mu}_1$ is not a behavioral strategy. No matter how Player 2 responds, Player 1 is then guaranteed to get a utility of $1/2$. On the other hand, let's suppose that Player 1 is restricted to only play behavioral strategies. Since Player 2 is minimizing the utility of Player 1, Player 1 will get a utility of

$$\begin{aligned}\min(\boldsymbol{b}_1[\mathsf{I}_1, \mathsf{L}_1] \cdot \boldsymbol{b}_1[\mathsf{I}_2, \mathsf{L}_2],\ \boldsymbol{b}_1[\mathsf{I}_1, \mathsf{R}_1] \cdot \boldsymbol{b}_1[\mathsf{I}_2, \mathsf{R}_2]) = \\ \min\left(\boldsymbol{b}_1[\mathsf{I}_1, \mathsf{L}_1] \cdot \boldsymbol{b}_1[\mathsf{I}_2, \mathsf{L}_2], (1 - \boldsymbol{b}_1[\mathsf{I}_1, \mathsf{L}_1]) \cdot (1 - \boldsymbol{b}_1[\mathsf{I}_2, \mathsf{L}_2])\right). \quad (2)\end{aligned}$$

It's not hard to see that the optimal solution of this expression is $1/4$, which is considerably lower than what is possible using mixed strategies—namely, $1/2$. But Theorem 1.2 reassures us that this cannot happen under perfect recall.

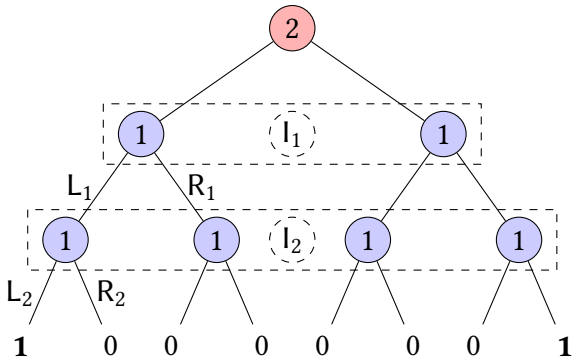

Figure 1: An imperfect-recall game where not correlating between information sets hurts the player's utility (*cf.* Theorem 1.2).

Sequence form While operating over behavioral strategies in perfect-recall games is as expressive as considering the entire set of mixed strategies (Theorem 1.2), working with behavioral strategies has a basic flaw from an optimization perspective: even when one fixes the strategies of the rest of the players, the utility of that player *is not a linear function*—not even concave—in that player's strategy, thereby failing to meet the second desideratum (Item 2) laid out earlier. (Item 1 is, by contrast, satisfied: the set of behavioral strategies is simply a Cartesian product of simplices, which is a convex polytope that can be described with a polynomial number of constraints.) This becomes evident already from the expression of the utility in (2). More broadly, similarly to (1), the utility as a function of a behavioral strategy profile $(\boldsymbol{b}_1, \dots, \boldsymbol{b}_n)$ can be expressed as

$$\sum_{z \in \mathcal{Z}} u_i(z) p_c(z) \prod_{i' \in [n]} \prod_{\substack{(h,a) \preceq z \\ h \in \mathcal{H}_{i'}}} \boldsymbol{b}_{i'}[(j,a)].$$

The basic issue is that, even after fixing all but one player, the utility function of that player will contain products, which are not easy to handle from an optimization standpoint. Thankfully, there is a neat trick that sidesteps this issue—the *sequence-form* representation of strategies.

The sequence form was discovered independently by Romanovskii [1962] and Koller et al. [1996]. A sequence-form vector, denoted by $\boldsymbol{x}_i$, again belongs to $\mathbb{R}^{\Sigma_i}$. But the key difference is that, for each sequence $(j,a) \in \Sigma_i$, the entry of $\boldsymbol{x}_i[(j,a)]$ contains the *product* of the probabilities on the path from the root to action a at decision point j. In particular, $\boldsymbol{x}_i[\sigma] \coloneqq \prod_{(j,a) \preceq \sigma} \boldsymbol{b}_i[(j,a)]$ for all sequences $\sigma \in \Sigma_i$, where $\boldsymbol{b}_i$ is a behavioral strategy. For $\boldsymbol{x}_i$ to be a valid sequence-form strategy, it is enough to ensure that the probability mass is conserved, in that $\sum_{a \in \mathcal{A}_j} \boldsymbol{x}_i[(j,a)] = \boldsymbol{x}_i[p_j]$, where p_j is i's sequence preceding j; if j is not preceded by some another sequence—in which case we write $p_j = \varnothing$—we instead have $\sum_{a \in \mathcal{A}_j} \boldsymbol{x}_i[(j,a)] = 1$.

Definition 1.3 (Sequence-form polytope)**.** The *sequence-form polytope* is the convex set

$$\mathcal{X}_i \coloneqq \left\{ \boldsymbol{x}_i \in \mathbb{R}^{\Sigma_i}_{\geq 0} : \sum_{a \in \mathcal{A}_j} \boldsymbol{x}_i[(j,a)] = \begin{cases} 1 & \text{if } p_j = \varnothing \\ \boldsymbol{x}_i[p_j] & \text{otherwise.} \end{cases} \quad \forall j \in \mathcal{J}_i \right\}.$$

In particular, the number of constraints of $\mathcal{X}_i$ is polynomial in the size of the game. What's more, the utility of each player i can now be expressed as $u_i(\boldsymbol{x}) = \sum_{z \in \mathcal{Z}} u_i(z) p_c(z) \prod_{i'=1}^{n} \boldsymbol{x}_{i'}[\sigma_{i'}(z)]$, where $\sigma_{i'}(z)$ is the last sequence of player i' from the path of the root to z. We see that $u_i(\boldsymbol{x})$ can be expressed as $\langle \boldsymbol{x}_i, \boldsymbol{u}_i(\boldsymbol{x}_{-i}) \rangle$ for some function $\boldsymbol{u}_i(\boldsymbol{x}_{-i}) \in \mathbb{R}^{\Sigma_i}$ that does not depend on $\boldsymbol{x}_i$.

Zero-sum games In the special case of zero-sum games, the utility of Player 2 reads $\boldsymbol{x}^\top \mathbf{A} \boldsymbol{y}$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are sequence-form vectors (per Definition 1.3); as in the previous lecture, we take Player 1—who selects $\boldsymbol{x}$—to be the minimizer. We will now show how to make use of the sequence-form representation to efficiently compute minimax equilibria in zero-sum extensive-form games.

By Definition 1.3, we can represent the sequence-form strategy polytope of Player 1 as $\mathcal{X} \coloneqq \{\boldsymbol{x} \in \mathbb{R}^{\Sigma_1}_{\geq 0} : \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1\}$, and similarly, $\mathcal{Y} \coloneqq \{\boldsymbol{x} \in \mathbb{R}^{\Sigma_2}_{\geq 0} : \mathbf{F}_2 \boldsymbol{y} = \boldsymbol{f}_2\}$. Let's fix the strategy of Player 1, $\boldsymbol{x} \in \mathcal{X}$, and consider the following linear program.

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{x}^\top \mathbf{A} \boldsymbol{y} \\ \text{subject to} \quad & \mathbf{F}_2 \boldsymbol{y} = \boldsymbol{f}_2, \\ & \boldsymbol{y} \geq 0. \end{aligned} \tag{P}$$

By strong duality, we know that for any $\boldsymbol{x}$, the value of (P) is equal to the value of its dual; namely,

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{subject to} \quad & \mathbf{A}^\top \boldsymbol{x} \geq \mathbf{F}_2^\top \boldsymbol{v}. \end{aligned} \tag{D}$$

In other words, we have shown that, for any $\boldsymbol{x} \in \mathcal{X}$, $\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{x}^\top \mathbf{A} \boldsymbol{y} = \min_{\boldsymbol{v}: \mathbf{A}^\top \boldsymbol{x} \geq \mathbf{F}_2^\top \boldsymbol{v}} \boldsymbol{f}_2^\top \boldsymbol{v}$. This implies that one can compute a solution to $\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{x}^\top \mathbf{A} \boldsymbol{y}$ by solving the following linear program with respect to the variables $(\boldsymbol{x}, \boldsymbol{v})$.

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{subject to} \quad & \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1, \\ & \boldsymbol{x} \geq 0, \\ & \mathbf{A}^\top \boldsymbol{x} \geq \mathbf{F}_2^\top \boldsymbol{v}. \end{aligned}$$

We thus arrive at the following theorem, which extends a result covered in the last lecture pertaining to normal-form zero-sum games.

Theorem 1.4. *For any perfect-recall zero-sum extensive-form game, a minimax equilibrium can be computed in time polynomial in the size of the game tree.*

2 Tree-form decision problems

We now introduce the framework of *tree-form decision making*. In a nutshell, it describes the decision problem faced by a single player in an extensive-form game when we abstract away all other players.

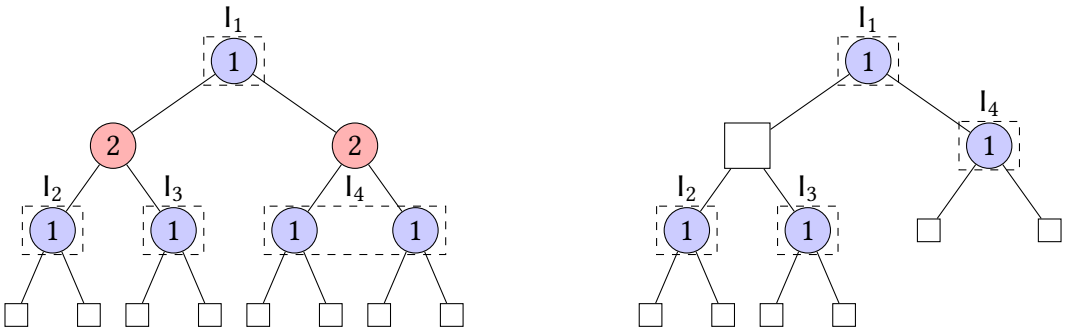

Figure 2: Left: a two-player extensive-form game. Right: the induced tree-form decision problem faced by Player 1. It contains 4 decision points and a single observation point.

Under this abstraction, the player interacts with an environment in two basic ways. In a decision point $j \in \mathcal{J}$, corresponding to an information set, the player is to select an action from a set $\mathcal{A}_j$. At an observation point $k \in \mathcal{K}$, the player observes a signal $s \in \mathcal{S}_k$. It is assumed that decision and observation points form a tree. Figure 2 gives a simple example showing how one can think about the decision problem faced by Player 1 in terms of a tree-form decision problem.

We further assume that there is a *transition function* ρ. Given a decision node $j \in \mathcal{J}$ and an action $a \in \mathcal{A}_j$, $\rho(j, a)$ returns the next point $p \in \mathcal{J} \cup \mathcal{K} \cup \{\perp\}$ in the tree reached upon selecting a in j, or $\perp$ if the decision process ends. Similarly, given an observation node $k \in \mathcal{K}$ and a signal $s \in \mathcal{S}_k$, $\rho(k, s)$ returns the next point $p \in \mathcal{J} \cup \mathcal{K} \cup \{\perp\}$ in the tree reached upon observing s in k, or $\perp$ if the decision process ends.

2.1 Inductive decomposition of strategies

Let $\mathcal{X}$ be the sequence-form strategy set in the underlying tree-form decision problem (per Definition 1.3). We will now see how to decompose $\mathcal{X}$ using two basic operations: convex hulls and Cartesian products. Let's start from the example of Figure 2. We proceed in a *bottom-up* fashion. At the terminal decision points $j = I_2, I_3, I_4$, the strategy set $\mathcal{X}_j$ is a probability simplex; namely, $\mathcal{X}_j = \Delta^2$. Next, at the only observation point, say k, we have $\mathcal{X}_k = \mathcal{X}_{I_2} \times \mathcal{X}_{I_3}$; that is, it can be decomposed into two independent subproblems. Finally, at I_1, the player needs to specify a probability for playing each of the two actions. Let's denote them by $\boldsymbol{\lambda}[a_1]$ and $\boldsymbol{\lambda}[a_2]$. By definition of the sequence-form polytope, one then has to multiply by $\boldsymbol{\lambda}[a_1]$ the strategies in the subtree rooted at k and by $\boldsymbol{\lambda}[a_2]$ the strategies in the subtree rooted at I_4. In other words, $\mathcal{X} \coloneqq \mathcal{X}_{I_1} = \{(\boldsymbol{\lambda}[a_1], \boldsymbol{\lambda}[a_2], \boldsymbol{\lambda}[a_1]\boldsymbol{x}_k, \boldsymbol{\lambda}[a_2]\boldsymbol{x}_{I_4}) : (\boldsymbol{\lambda}[a_1], \boldsymbol{\lambda}[a_2]) \in \Delta^2, \boldsymbol{x}_k \in \mathcal{X}_k, \boldsymbol{x}_{I_4} \in \mathcal{X}_{I_4}\}$.

There is a direct way of extending this bottom-up decomposition of the sequence-form polytope in general tree-form decision problems. First, in any observation point $k \in \mathcal{K}$ with a set of children $\{p_1, \dots, p_v\} = \{\rho(k, s) : s \in \mathcal{S}_k\} \eqqcolon C_k$, we have

$$\mathcal{X}_k = \mathcal{X}_{p_1} \times \mathcal{X}_{p_2} \times \cdots \times \mathcal{X}_{p_v},$$

where $\mathcal{X}_{p_i}$ is the sequence-form polytope corresponding to the subtree rooted at point p_i. In any

decision point $j \in \mathcal{J}$ with a set of children $\{p_1, \dots, p_v\} = \{\rho(j, a) : a \in \mathcal{A}_j\} =: C_j$,

$$\mathcal{X}_j \coloneqq \text{co}\left\{\begin{pmatrix} \boldsymbol{e}_1 \\ \mathcal{X}_{p_1} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix}, \begin{pmatrix} \boldsymbol{e}_2 \\ \mathbf{0} \\ \mathcal{X}_{p_2} \\ \vdots \\ \mathbf{0} \end{pmatrix}, \dots, \begin{pmatrix} \boldsymbol{e}_v \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathcal{X}_{p_v} \end{pmatrix}\right\},$$

where $\boldsymbol{e}_i \in \mathbb{R}^{\mathcal{A}_j}$ is the ith unit vector.

3 Regret circuits and counterfactual regret minimization

We will now leverage the decomposition of Section 2.1 to construct a regret minimizer for the sequence-form polytope. This section follows the framework of *regret circuits*, introduced by Farina et al. [2019]. The resulting algorithm—in fact, family of algorithms—is CFR.

Regret circuits address the following basic question. Let's say we know how to efficiently minimize regret over a certain set. Can one *compose* multiple such regret minimizers to tackle a composite, more complex set? Last lecture gave multiple efficient no-regret algorithms for the probability simplex. Further, Section 2.1 shows that one can express the sequence-form polytope as a composite set of probability simplices involving only i) Cartesian products and ii) convex hulls. As a result, it will be enough to design regret circuits for those two basic operations.

Remark 3.1. Before we proceed, we remark that while the previous lecture introduced the framework of regret minimization in the special case where the learner outputs a probability distribution, the definitions readily carry over when the learner outputs a point $\boldsymbol{x}$ in a general convex set $\mathcal{X}$ (for example, the sequence-form polytope), and then obtains a linear utility function of the form $\boldsymbol{x} \mapsto \langle \boldsymbol{x}, \boldsymbol{u} \rangle$.

3.1 Cartesian product

Let's say we have an online algorithm $\mathfrak{R}_p$ that efficiently minimizes regret over a set $\mathcal{X}_p$ for each $p \in C_k$. We will now develop a regret minimizer $\mathfrak{R}_k$ for $\mathcal{X}_k \coloneqq \bigtimes_{p \in C_k} \mathcal{X}_p$. Recall from the last lecture that an online algorithm interacts with the environment by first specifying a strategy in $\mathcal{X}_k$. Then it observes a utility feedback and can update the internal state of the algorithm accordingly. We will handle those two separately.

- Any time $\mathfrak{R}_k$ needs to specify a strategy, it obtains the strategy $\boldsymbol{x}_p$ of $\mathfrak{R}_p$ for all $p \in C_k$, and then outputs the Cartesian product $(\boldsymbol{x}_p)_{p \in C_k}$.
- Any time $\mathfrak{R}_k$ obtains a utility vector $\boldsymbol{u}_k = (\boldsymbol{u}_p)_{p \in C_k}$ as feedback from the environment, it forwards $\boldsymbol{u}_p$ to each $\mathfrak{R}_p$.

It is straightforward to show that the regret of $\mathfrak{R}_k$ is the sum of the regrets of $(\mathfrak{R}_p)_{p \in C_k}$; the proof is left as a simple exercise.

Proposition 3.2. *The regret of $\mathfrak{R}_k$ is equal to $\sum_{p \in C_k} \mathrm{Reg}_p^{(T)}$, where $\mathrm{Reg}_p^{(T)}$ is the regret of $\mathfrak{R}_p$.*

In particular, if the regret of each $\mathfrak{R}_p$ grows sublinearly in T, so does the regret of $\mathfrak{R}_k$.

3.2 Convex hull

The convex hull requires a slightly more involved construction. Here, we assume that we have a regret minimizer $\mathfrak{R}_p$ with respect to a set C_j, for each $p \in C_j$, and the goal is to produce a regret minimizer $\mathfrak{R}_j$ for the set $\mathcal{X}_j \coloneqq \mathrm{co}\{(\mathcal{X}_p)_{p \in C_j}\}$. We will make use of a regret minimizer $\mathfrak{R}_\Delta$ that operates over $\Delta(C_j)$; the basic role of $\mathfrak{R}_\Delta$ is to specify how to mix over $(\mathfrak{R}_p)_{p \in C_j}$. As before, there is a natural algorithm for implementing a regret circuit for the convex hull.

- Any time $\mathfrak{R}_j$ needs to specify a strategy, it obtains the strategy $\boldsymbol{x}_p$ of $\mathfrak{R}_p$, for all $p \in C_j$, and the strategy $\boldsymbol{\lambda} \in \Delta(C_j)$ of $\mathfrak{R}_\Delta$, and then outputs $(\boldsymbol{\lambda}[p]\boldsymbol{x}_p)_{p \in C_j}$.
- Any time $\mathfrak{R}_j$ obtains a utility vector $\boldsymbol{u}_j$ as feedback from the environment, it first forwards $\boldsymbol{u}_j$ to each $\mathfrak{R}_p$. It then forwards to $\mathfrak{R}_\Delta$ the utility vector $(\langle \boldsymbol{x}_p, \boldsymbol{u}_j \rangle)_{p \in C_j}$.

It's not hard to show that the above construction yields a sound regret circuit for the convex hull; the proof is left again as an exercise.

Proposition 3.3. *The regret of $\mathfrak{R}_j$ is at most $\mathrm{Reg}_\Delta^{(T)} + \max_{p \in C_j} \mathrm{Reg}_p^{(T)}$, where $\mathrm{Reg}_\Delta^{(T)}$ is the regret of $\mathfrak{R}_\Delta$ and $\mathrm{Reg}_p^{(T)}$ is the regret of each $\mathfrak{R}_p$.*

Combining Propositions 3.2 and 3.3 with the decomposition of Section 2.1, together with the fact that efficient regret minimizers over the simplex exist, we obtain an efficient online algorithm for minimizing regret over the sequence-form polytope.

Theorem 3.4. *There exists an efficient algorithm for minimizing regret with respect to the sequence-form polytope.*

3.3 Counterfactual regret minimization

We spell out the resulting construction in Algorithm 1. This algorithm was first proposed and analyzed by Zinkevich et al. [2007], although the preceding analysis follows the framework of Farina et al. [2019]. `CFR` can be thought of as a family of algorithms: it can be instantiated with any regret minimizer over the simplex. By far the most common choice in practice is to use regret matching and its modern variants, to be discussed more in the next lecture.

In Algorithm 1, we denote by NextStrategy the function that outputs the next strategy of `CFR`. It invokes each local regret minimizer, and it then proceeds by converting the resulting behavioral strategy into a sequence-form vector; this conversion is essential for reasons we have already discussed. The function ObserveUtility updates the so-called *counterfactual utilities* that are given as input to the local regret minimizers that operate over each decision point.

Algorithm 1: Counterfactual regret minimization (CFR)

Input: A regret minimizer $\mathfrak{R}_j$ for each decision point $j \in \mathcal{J}$ of the TFDP.
NextStrategy():
  for *each decision point* $j \in \mathcal{J}$ **do**
    $\Delta(\mathcal{A}_j) \ni b_j^{(t)} \coloneqq \mathfrak{R}_j$.NextStrategy();
  for *each decision point* $j \in \mathcal{J}$ *in top-down order* **do**
    for *each action* $a \in \mathcal{A}_j$ **do**
      if $p_j = \emptyset$ **then**
        $\boldsymbol{x}^{(t)}[(j,a)] \coloneqq b_j^{(t)}[a]$;
      else
        $\boldsymbol{x}^{(t)}[(j,a)] \coloneqq \boldsymbol{x}^{(t)}[p_j] \cdot b_j^{(t)}[a]$;
  return $\boldsymbol{x}^{(t)} \in \mathbb{R}^{\Sigma}$;
ObserveUtility($\boldsymbol{u}^{(t)} \in \mathbb{R}^{\Sigma}$):
  $V^{(t)}[\perp] \coloneqq 0$;
  for *each node in the tree* $p \in \mathcal{J} \cup \mathcal{K}$ *in bottom-up order* **do**
    if $p \in \mathcal{J}$ **then**
      Let $j = p$;
      $V^{(t)}[j] \coloneqq \sum_{a \in \mathcal{A}_j} b_j^{(t)}[a] \cdot (\boldsymbol{u}^{(t)}[(j,a)] + V^{(t)}[\rho(j,a)])$;
    else
      Let $k = p$;
      $V^{(t)}[k] \coloneqq \sum_{s \in \mathcal{S}_k} V^{(t)}[\rho(k,s)]$;
  for *each decision point* $j \in \mathcal{J}$ **do**
    for *each action* $a \in \mathcal{A}_j$ **do**
      $\boldsymbol{u}_j^{(t)}[a] \coloneqq \boldsymbol{u}^{(t)}[(j,a)] + V^{(t)}[\rho(j,a)]$;
    $\mathfrak{R}_j$.ObserveUtility($\boldsymbol{u}_j^{(t)}$);

References

H. W. Kuhn. Extensive games and the problem of information. In *Contributions to the Theory of Games*, volume 2 of *Annals of Mathematics Studies, 28*, pages 193–216. Princeton University Press, 1953.

I. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3, 1962.

Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2), 1996.

Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Regret circuits: Composability of regret minimizers. In *International Conference on Machine Learning (ICML)*, 2019.

Martin Zinkevich, Michael Bowling, Michael Johanson, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Neural Information Processing Systems (NIPS)*, 2007.