Lecture Note 6: Extensive-Form Games

Christian Kroer[*]

February 6, 2022

1 Introduction

In this lecture we will cover *extensive-form games* (EFGs). Extensive-form games are a richer game description that explicitly models sequential interaction. EFGs are played on a game tree. Each node in the game tree belongs to some player, whom gets to choose the branch to traverse.

2 Perfect-Information EFGs

We start by considering *perfect-information* EFGs. The term perfect information refers to the fact that in these games, every player always knows the exact state of the game. A perfect-information EFG is a game played on a tree, where each internal node belongs to some player. The actions for the player at a given node is the set of branches, and by selecting a branch the game proceeds to the following node. An example is shown in Figure 1 on the left. That game has four nodes where players take actions, two belong to player 1 (labelled P1) and two belonging to player 2 (labelled P2). Additionally, the game tree has 6 leaf nodes. At each leaf node, each player receives some payoff. In this particular game, it is a zero-sum game, and the value at a leaf denotes the value that player 1 receives.

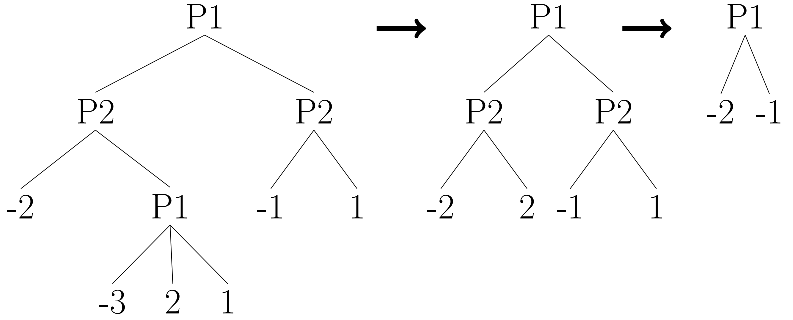

Figure 1: A simple perfect-information EFG. Three versions of the game are shown, where each stage corresponds to removing one layer of the game via backward induction.

Perfect-information EFGs are trivially solvable (at least if we are able to traverse the whole game tree at least once). The way to solve them is via *backward induction*. Backward induction works by starting at some bottom decision node of the game tree, which only leads to leaf nodes

[*] Department of Industrial Engineering and Operations Research, Columbia University. Email: christian.kroer@columbia.edu.

after each action is taken (such a node always exists). Then, the optimal action for the player at the node is selected, and the node is replaced with the corresponding leaf node. Now we get a new perfect-information EFG with one less internal node. Backward induction then repeats this process until there's no internal nodes left, at which point we have computed a Nash equilibrium. Thus perfect-information EFGs always have pure-strategy Nash equilibria.

While backward induction yields a linear-time algorithm for solving perfect-information games, in practice, many games of interest are way too large to solve with it nonetheless. For example, chess and go both have enormous game trees, with estimates of $\sim 10^{45}$ and $\sim 10^{172}$ nodes respectively.

Next let us see how converting to normal form works. The way converting to normal form works is that for each player, we create an action corresponding to every possible way of assigning an action at every decision point. So, if a player has d decision points with A actions each, then there A^d actions will be created in the normal form representation of the EFG. This reduction to normal form works for both perfect and imperfect-information games.

Let's consider an instructive example. Here we will model the Cuban Missile Crisis. The USSR has moved a bunch of nuclear weapons to Cuba, and the US has to decide how to respond. If they do nothing, then the USSR wins a political victory, and gets to keep nuclear missiles within firing distance of major US cities. If the US responds, then it could results in a series of escalations that would eventually lead to nuclear war, or the USSR will eventually compromise and remove the missiles.

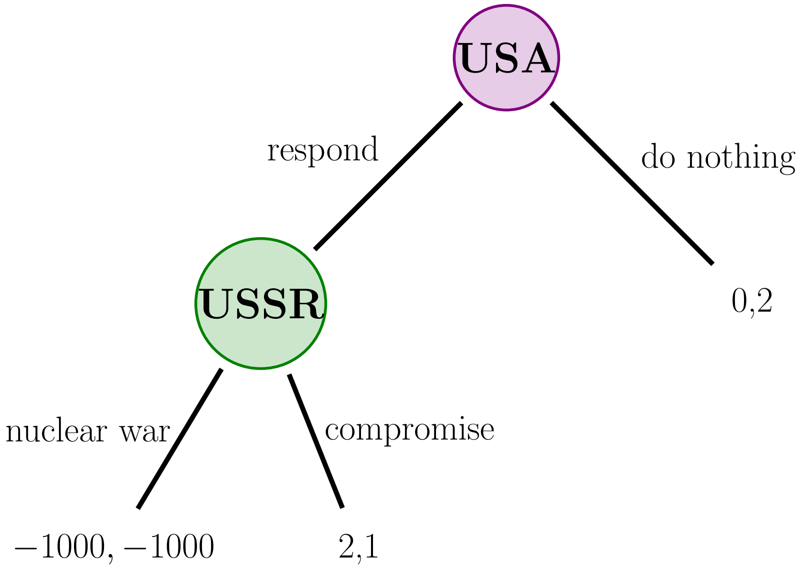

Figure 2: A perfect-information EFG modeling the Cuban missile crisis.

If we convert this game to normal form, we get the following game:

		USSR	
		Nuclear war	Compromise
USA	Respond	$-1000, -1000$	2,1
	Do Nothing	0,2	0,2

It is straightforward to see from this representation that the Cuban Missile Crisis game has two PNE: (do nothing, nuclear war) and (respond, compromise). However, the first PNE is in a sense not compelling: what if the USA just responded? The USSR probably would not be willing to follow through on taking the action “nuclear war” since it has such low utility for them as well. This leads to the notion of *subgame-perfect equilibria*, which are equlibria that remain equilibria if we take any *subgame* consisting of picking some node in the tree and starting the game there.

3 Imperfect-Information EFGs

An example is shown in Figure 3.

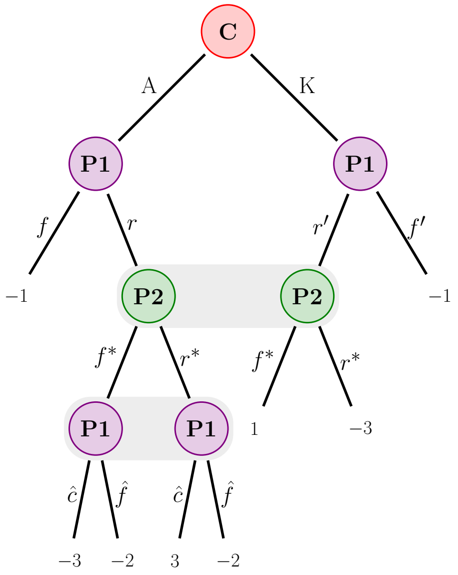

Figure 3: A (rather weird) poker game where P1 is dealt Ace or King with equal probability. “r,” “f,” and “c” stands for raise, fold, and check respectively. Leaf values denote P1 payoffs. The shaded area denotes an information set: P2 does not know which of these nodes they are at, and must thus use the same strategy in both. Note that in the case where they are dealt an ace, P1 does not observe the action taken by P2.

An EFG has the following:

- Information sets: for each player, the nodes belonging to that player are partitioned into *information sets* $I \in \mathcal{I}_i$. information sets represent imperfect information: a player does not know which node in an information set they are at, and thus they muse utilize the same strategy at each node in that information set. In Figure 3 P2 has only 1 information set, which contains both their nodes, whereas P1 has four information sets, each one a singleton node. For player i we will also let $\mathcal{J}_i$ be an index set of information sets with generic element j.

- Each information set I with index j has a set of actions that they corresponding player may take, which is denoted by A_j.

- Leaf nodes Z: the set of terminal states. Player i gains utility $u_i(z)$ if leaf node z is reached. Z is the set of all leaf nodes.

- Chance nodes where Chance or Nature moves with a fixed probability distribution. In Figure 3 chance deals A or K with equal probability.

We will assume throughout that the game has *perfect recall*, which means that no player ever forgets something they knew in the past. More formally, it means that for every information set $I \in \mathcal{I}_i$, there is a single last information-set action pair I', a' belonging to i that was the last information set and action taken by that player for every node in I.

The last action taken by player i before reaching an information set with index j is denoted p_j. This is well-defined due to perfect recall.

We spent a lot of time learning how one may compute a Nash equilibrium in a two-player zero-sum game by finding a saddle point of a min-max problem over convex compact polytopes. This model looked as follows (we also learned how to handle convex-concave objectives, here we restrict our attention to bilinear saddle-point problems)

$$\min_{x \in X} \max_{y \in Y} \langle x, Ay \rangle. \tag{1}$$

Now we would like to find a way to represent EFG zero-sum Nash equilibrium this way. This turns out to be possible, and the key is to find the right way to represent strategies such that we get a bilinear objective. The next section will describe this representation.

First, let us see why the most natural formulation of the strategy spaces won't work. The natural formulation would be to have a player specify a probability distribution over actions at each of their information sets. Let σ be a strategy profile, where σ_a is the probability of taking action a (from now on we assume that every action is distinct so that for any a there is only one corresponding I where the action can be played). The expected value over leaf nodes is

$$\sum_{z \in Z} u_2(z) \mathbb{P}(z|\sigma)$$

The problem with this formulation is that if a player has more than one action on the path to any leaf, then the probability $\mathbb{P}(z|\sigma)$ of reaching z is non-convex in that player's own strategy, since we have to multiply each of the probabilities belonging to that player on the path to z. Thus we cannot get the bilinear form in (1).

4 Sequence Form

In this section we will describe how we can derive a bilinear representation X of the strategy space for player 1. Everything is analogous for Y.

In order to get a bilinear formulation of the expected value we do not write our strategy in terms of the probability σ_a of playing an action a. Instead, we associate to each information-set-action pair I, a a variable x_a denoting the probability of playing the *sequence* of actions belonging to player 1 on the path to I, including the probability of a at I. For example, in the poker game in Figure 3, there would be a variable $x_{\hat{c}}$ denoting the product of probabilities player 1 puts on playing actions r and then $\hat{c}$. To be concrete, say that we have a behavioral strategy σ^1 for player 1, then the corresponding sequence-form probability on the action $\hat{c}$ would be $x_{\hat{c}} = \sigma^1_r \cdot \sigma^1_{\hat{c}}$. Similarly there would be a variable $x_{\hat{f}} = \sigma^1_r \cdot \sigma^1_{\hat{f}}$ denoting the product of probabilities on r and $\hat{f}$. Clearly, for this to define a valid strategy we must have $x_{\hat{c}} + x_{\hat{f}} = x_r$.

More generally, X is defined as the set of all $x \in \mathbb{R}^n, x \geq 0$ such that

$$x_{p_j} = \sum_{a \in A_j} x_a, \forall j \in \mathcal{J}_1, \tag{2}$$

where $n = \sum_{I \in \mathcal{I}_i} |A|$, and $p(I)$ is the parent sequence leading to I.

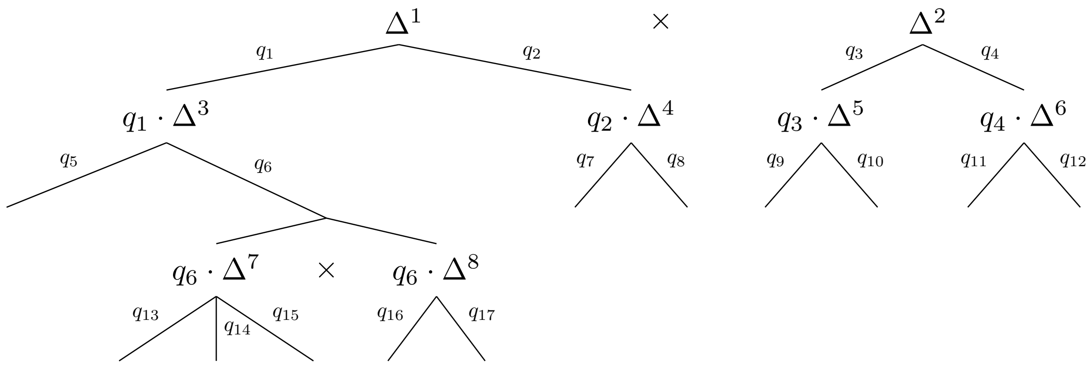

Figure 4: An example treeplex constructed from 9 simplices.

One way to visually think of the set of sequence-form strategies is given in Figure 4. This representation is called a *treeplex*. Each information set is represented as a simplex, which is scaled by the parent sequence leading to that information set (by perfect recall there is a unique parent sequence). After taking a particular action it is possible that a player may arrive at several next possible simplexes depending on what the other player or nature does. This is represented by the × symbol.

It's important to understand that the sequence form specifies probabilities on sequences of actions *for a single player*. Thus they are not the same as paths in the game tree; indeed, the sequence r^* for player 2 appears in two separate paths of the game tree, as player 2 has two nodes in the corresponding information set.

Say we have a set of probability distributions over actions at each information set, with σ_a denoting the probability of playing action a. We may construct a corresponding sequence-form strategy by applying the following equation in top-down fashion (so that x_{p_j} is always assigned before x_a):

$$x_a = x_{p_j}\sigma_a, \forall j \in \mathcal{J}, a \in A_j. \tag{3}$$

The payoff matrix A associated with the sequence-form setup is a sparse matrix, with each row corresponding to a sequence of the x player and each column corresponding to a sequence of the y player. Each leaf has a cell in A at the pair of sequences that are last visited by each player before reaching that leaf, and the value in the cell is the payoff to the maximizing player in the bilinear min-max formulation. Cells corresponding to pairs of sequences that are never the last pair of sequences visited before a leaf have a zero.

With this setup we now have an algorithm for computing a Nash equilibrium in a zero-sum EFG: Run online mirror descent (OMD) for each player, using either of our folk-theorem setups from the previous lecture. However, this has one issue, recall the update for OMD (also known as a *prox mapping*):

$$x_{t+1} = \operatorname*{argmin}_{x \in X} \langle \gamma g_t, x \rangle + D(x \| x_t),$$

where $D(x\|x_t) = d(x) - d(x_t) - \langle \nabla d(x_t), x - x_t \rangle$ is the Bregman divergence from x_t to x. In order to run OMD, we need to be able to compute this prox mapping. The question of whether the prox mapping is easy to compute is easily answered when X is a simplex, where updates for the entropy DGF are closed-form, and updates for the Euclidean DGF can be computed in $n \log n$ time, where n is the number of actions. For treeplexes this question becomes more complicated.

In principle we could use the standard Euclidean distance for d. In that case the update can be rewritten as

$$x_{t+1} = \operatorname*{argmin}_{x\in X} \|x - (x_t - \gamma g_t)\|_2^2,$$

which means that the update requires us to project onto a treeplex. This can be done in $n \log n$ time, where n is the number of sequences. While this is acceptable, it turns out there are smarter ways to compute these updates which take linear time in n.

5 Dilated Distance-Generating Functions

We will see two ways to construct regret minimizers for treeplexes. The first is based on choosing an appropriate distance-generating function (DGF) for the treeplex, such that prox mappings are easy to compute. To that end, we now introduce what are called *dilated DGFs*. In dilated DGFs we assume that we have a DGF d_j for each information set $j \in \mathcal{J}$. For the polytope X we construct the DGF

$$d(x) = \sum_{j\in\mathcal{J}_1} \beta_j x_{p_j} d_j\left(\frac{x^j}{x_{p_j}}\right),$$

where $\beta_j > 0$ is the weight on information set j.

Dilated DGFs have the nice property that the proximal update can be computed recursively as long as we know how to compute the simplex update for each j. Let x^j, g_t^j etc denote the slice of a given vector corresponding to sequences belonging to information set j. The update is

$$\begin{aligned}
&\operatorname*{argmin}_{x\in X} \langle g_t, x\rangle + D(x\|x_t)\\
=&\operatorname*{argmin}_{x\in X} \langle g_t, x\rangle + d(x) - d(x_t) - \langle \nabla d(x_t), x - x_t\rangle\\
=&\operatorname*{argmin}_{x\in X} \langle g_t - \nabla d(x_t), x\rangle + d(x)\\
=&\operatorname*{argmin}_{x\in X} \sum_{j\in\mathcal{J}} \Big(\langle g_t^j - \nabla d(x_t)^j, x^j\rangle + \beta_j x_{p_j} d_j(x^j/x_{p_j})\Big)\\
=&\operatorname*{argmin}_{x\in X} \sum_{j\in\mathcal{J}} x_{p_j} \Big(\langle g_t^j - \nabla d(x_t)^j, x^j/x_{p_j}\rangle + \beta_j d_j(x^j/x_{p_j})\Big)
\end{aligned}$$

Now we may consider some information set j with no descendant information sets. Since x_{p_j} is on the outside of the parentheses, we can compute the update at j as if it were a simplex update, and the value at the information set can be added to the coefficient on x_{p_j}. That logic can then be applied recursively. Thus we can traverse the treeplex in bottom-up order, and at each information set we can compute the value for x_{t+1}^j in however long it takes to compute an update for a simplex with DGF d_j.

If we use the entropy DGF for each $j \in \mathcal{J}$ and set the weight $\beta_j = 2 + \max_{a\in A_j} \sum_{j'\in\mathcal{C}_j^a} 2\beta_{j'}$, then we get a DGF for X that is strongly convex modulus $\frac{1}{M}$ where $M = \max_{x\in X} \|x\|_1$. If we scale this DGF by M we get that it is strongly convex modulus 1. If we instantiate the mirror prox algorithm with this DGF for X and Y we get an algorithm that converges at a rate of

$$O\left(\frac{\max_{ij} A_{ij} \max_{I\in\mathcal{I}} \log(|A_I|)\sqrt{M_x^2 2^d + M_y^2 2^d}}{T}\right),$$

where M_x, M_y are the maximum ℓ_1 norms on X and Y, and d is an upper bound on the depth of both treeplexes. This gives the fastest theoretical rate of convergence among gradient-based methods. However, this only works for OMD. All our other algorithms (RM, RM$^+$) were for simplex domains exclusively. Next we derive a way to use these locally at each information set. It turns out that faster practical performance can be obtained this way.

6 Counterfactual Regret Minimization

The framework we will cover is the *counterfactual regret minimization* (CFR) framework for constructing regret minimizers for EFGs.

CFR is based on deriving an upper bound on regret, which allows decomposition into local regret minimization at each information set.

We are interested in minimizing the standard regret notion over the sequence form:

$$R_T = \sum_{t=1}^{T} \langle g_t, x_t \rangle - \min_{x \in X} \sum_{t=1}^{T} \langle g_t, x \rangle.$$

To get the decomposition, we will define a local notion of regret which is defined with respect to behavioral strategies $\sigma \in \times_j \Delta^j =: \Sigma$ (here we just derive the decomposition for a single player, say player 1. Everything is analogous for player 2).

We saw in the previous lecture note that it is always possible to go from behavioral form to sequence form using the following recurrence, where assignment is performed in top-down order.

$$x_a = x_{p_j} \sigma_a, \forall j \in \mathcal{J}, a \in A_j. \tag{4}$$

It is also possible to go the other direction (though this direction is not a unique mapping, as one has a choice of how to assign behavioral probabilities at information sets j such that $x_{p_j} = 0$). These procedures produce payoff-equivalent strategies for EFGs.

For a behavioral strategy vector σ (or loss vector g_t) we say that σ^j is the slice of σ corresponding to information set j. $\sigma^{j\downarrow}$ is the slice corresponding to j, *and every information set below* j. Similarly, $\Sigma^{j\downarrow}$ is the set of all behavioral strategy assignments for the subset of simplexes that are in the tree of simplexes rooted at j.

We let $\mathcal{C}_{j,a}$ be the set of next information sets belonging to player 1 that can be reached from j when taking action a. In other words, the set of information sets whose parent sequence is a.

Now, let the *value function* at time t for an information set j belonging to player 1 be defined as

$$V_t^j(\sigma) = \langle g_t^j, \sigma^j \rangle + \sum_{a \in A_j} \sum_{j' \in \mathcal{C}_{j,a}} \sigma_a V_t^{j'}(\sigma^{j'\downarrow}).$$

where $\sigma \in \Sigma^{j\downarrow}$. Intuitively, this value function represents the value that player 1 derives from information set j, assuming that i played to reach it, i.e. if we counterfactually set $x_{p_j} = 1$.

The *subtree regret* at a given information set j is

$$R_T^{j\downarrow} = \sum_{t=1}^{T} V_t^j(\sigma_t^{j\downarrow}) - \min_{\sigma \in \Sigma^{j\downarrow}} \sum_{t=1}^{T} V_t^j(\sigma),$$

Note that this regret is with respect to the behavioral form.

The local loss that we will eventually minimize is defined as

$$\hat{g}_{t,a} = g_{t,a} + \sum_{j' \in \mathcal{C}_{j,a}} V_t^{j'}(\sigma_t^{j'\downarrow}).$$

Note that for each j, the loss depends linearly on σ^j; σ^j does not affect information sets below j, since we use σ_t in the value function for child information sets j'.

Now we show that the subtree regret decomposes in terms of local losses and subtree regrets.

Theorem 1. *For any $j \in \mathcal{J}$, the subtree regret at time T satisfies*

$$R_T^{j\downarrow} = \sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma_t^j \rangle - \min_{\sigma \in \Delta^j} \left(\sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma \rangle - \sum_{a \in A_j, j' \in \mathcal{C}_{j,a}} \sigma_a R_T^{j'\downarrow} \right).$$

Proof. Using the definition of subtree regret we get

$$\begin{aligned} R_t^{j\downarrow} &= \sum_{t=1}^{T} V_t^j(\sigma_t^{j\downarrow}) - \min_{\sigma \in \Sigma^{j\downarrow}} \left(\sum_{t=1}^{T} \langle g_t^j, \sigma^j \rangle + \sum_{a \in A_j, j' \in \mathcal{C}_{j,a}} \sigma_a V_t^{j'}(\sigma^{j'\downarrow}) \right) && \text{by expanding } V_t^j(\sigma^{j\downarrow}) \\ &= \sum_{t=1}^{T} V_t^j(\sigma_t^{j\downarrow}) - \min_{\sigma \in \Delta^j} \left(\sum_{t=1}^{T} \langle g_t^j, \sigma \rangle + \sum_{a \in A_j, j' \in \mathcal{C}_{j,a}} \sigma_a \min_{\hat{\sigma} \in \Sigma^{j'\downarrow}} V_t^{j'}(\hat{\sigma}^{j'\downarrow}) \right) && \text{by sequential min} \\ &= \sum_{t=1}^{T} V_t^j(\sigma_t^{j\downarrow}) - \min_{\sigma \in \Delta^j} \left(\sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma \rangle - \sum_{a \in A_j, j' \in \mathcal{C}_{j,a}} \sigma_a R_T^{j'\downarrow} \right) && \text{by definition of } \hat{g}_t \text{ and } R_T^{j'\downarrow}. \end{aligned}$$

The theorem follows, since $V_t^j(\sigma_t^{j\downarrow}) = \langle \hat{g}_t^j, \sigma_t^j \rangle$. □

The local regret that we will be minimizing is the following

$$\hat{R}_T^j := \sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma_t^j \rangle - \min_{\sigma \in \Delta^j} \sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma \rangle.$$

Note that this regret is in the behavioral form, and it corresponds exactly to the regret associated to locally minimizing $\hat{g}_t^j$ at each simplex j.

The CFR framework is based on the following theorem, which says that the sequence-form regret can be upper-bounded by the behavioral-form local regrets.

Theorem 2. *The regret at time T satisfies*

$$R_T = R_T^{root\downarrow} \leq \max_{x \in X} \sum_{j \in \mathcal{J}} x_{p_j} \hat{R}_T^j,$$

where root is the root information set.

Proof. For the equality, consider the regret R_T over the sequence form polytope X. Since each sequence-form strategy has a payoff equivalent behavioral strategy in Σ and vice versa, we get that the regret R_T is equal to $R_T^{root\downarrow}$ for the root information set *root* (we may assume WLOG. that there is a root information set since if not then we can add a dummy root information set with a single action).

By Theorem 1 we have for any $j \in \mathcal{J}$

$$\begin{aligned} R_T^{j\downarrow} &= \sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma_t^j \rangle - \min_{\sigma \in \Delta^j} \left(\sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma \rangle - \sum_{a \in A_j, j' \in \mathcal{C}_{j,a}} \sigma_a R_T^{j'\downarrow} \right) \\ &\leq \sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma_t^j \rangle - \min_{\sigma \in \Delta^j} \sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma \rangle + \max_{\sigma \in \Delta^j} \sum_{a \in A_j, j' \in \mathcal{C}_{j,a}} \sigma_a R_T^{j'\downarrow}, \end{aligned} \tag{5}$$

where the inequality is by the fact that independently minimizing the terms $\sum_{t=1}^{T} \langle \hat{g}_t^j, \sigma \rangle$ and $-\sum_{a \in A_j, j' \in \mathcal{C}_{j,a}} \sigma_a R_T^{j'\downarrow}$ is smaller than jointly minimizing them.

Now we may apply (5) recursively in top-down fashion starting at *root* to get the theorem. □

A direct corollary of Theorem 2 is that if the counterfactual regret at each information set grows sublinearly then overall regret grows sublinearly. This is the foundation of the *counterfactual regret minimization* (CFR) framework for minimizing regret over treeplexes. The CFR framework can succinctly be described as

1. Instantiate a local regret minimizer for each information set simplex Δ^j.
2. At iteration t, for each $j \in \mathcal{J}$, feed the local regret minimizer the counterfactual regret $\hat{g}_t^j$.
3. Generate x_{t+1} as follows: ask for the next recommendation from each local regret minimizer. This yields a set of simplex strategies, one for each information set. Construct x_{t+1} via (4).

Thus we get an algorithm for minimizing regret on treeplexes based on minimizing counterfactual regrets. In order to construct an algorithm for computing a Nash equilibrium based on a CFR setup, we may invoke the folk theorems from the previous lectures using the sequence-form strategies generated by CFR. Doing this yields an algorithm that converges to a Nash equilibrium of an EFG at a rate on the order of $O\left(\frac{1}{\sqrt{T}}\right)$

While CFR is technically a framework for constructing local regret minimizers, the term “CFR” is often overloaded to mean the algorithm that comes from using the folk theorem with uniform averages, and using regret matching as the local regret minimizer at each information set. CFR$^+$ is the algorithm resulting from using the alternation setup, taking linear averages of strategies, and using RM$^+$ as the local regret minimizer at each information set.

We now show pseudocode for implementing the CFR algorithm with the RM$^+$ regret minimizer. In order to compute Nash equilibria with this method one would use CFR as the regret minimizer in one of the folk-theorem setups from the previous lecture.

Algorithm 1: CFR(RM$^+$)($\mathcal{J}, X$)

```
Data: $\mathcal{J}$ set of infosets
      $X$ sequence-form strategy space
function SETUP()
    $\mathbf{Q} \leftarrow$ 0-initialized vector over sequences
    $t \leftarrow 1$

function NEXTSTRATEGY()
    $x \leftarrow 0 \in \mathbb{R}^{|X|}$
    $x_\emptyset \leftarrow 1$
    for $j \in \mathcal{J}$ in top-down order do
        $s \leftarrow \sum_{a \in A_j} \mathbf{Q}_a$
        if $s = 0$ then
            for $a \in A_j$ do
                $x_a \leftarrow x_{p_j} \,/\, |A_j|$
        else
            for $a \in A_j$ do
                $x_a \leftarrow x_{p_j} \times \mathbf{Q}_a \,/\, s$
    return $x$

function OBSERVELOSS($g_t \in \mathbb{R}^{|X|}$)
    for $j \in \mathcal{J}$ in bottom-up order do
        $s \leftarrow \sum_{a \in A_j} \mathbf{Q}_a$
        $v \leftarrow 0$    // the value of information set $j$
        if $s = 0$ then
            $v \leftarrow \sum_{a \in A_j} g_{t,a}/|A_j|$
        else
            $v \leftarrow \sum_{a \in A_j} \langle g_{t,a}, \mathbf{Q}_a \,/\, s\rangle$
        $g_{t,p_j} \leftarrow g_{t,p_j} + v$    // construct local loss $\hat{g}_t$
        for $a \in A_j$ do
            $\mathbf{Q}_a \leftarrow [\mathbf{Q}_a + (v - g_{t,a})]^+$    // $g_{t,a} = \hat{g}_{t,a}$ since all $j' \in \mathcal{C}_{j,a}$ were already traversed
    $t \leftarrow t + 1$
```

NextStrategy simply implements the top-down recursion (4) while computing the update corresponding to RM$^+$ at each j. ObserveLoss uses bottom-up recursion to keep track of the regret-like sequence Q_a, which is based on $\hat{g}_{t,a}$ in CFR.

A technical note here is that we assume that there is some dummy sequence $\emptyset$ at the root of the treeplex with no corresponding j (this corresponds to a single-action dummy information set at the root, but leaving out that dummy information set in the index set $\mathcal{J}$). This makes code much cleaner because there is no need to worry about the special case where a given j has no parent sequence, at the low cost of increasing the length of the sequence-form vectors by 1.

7 Numerical Comparison of CFR methods and OMD-like methods

Figure 5 shows the performance of three different variations of CFR, as well as the *excessive gap technique* (EGT), a first-order method that converges at a rate of $O(1/T)$ using the dilated entropy DGF from last lecture (EGT is equivalent to the mirror prox algorithm that was shown previously,

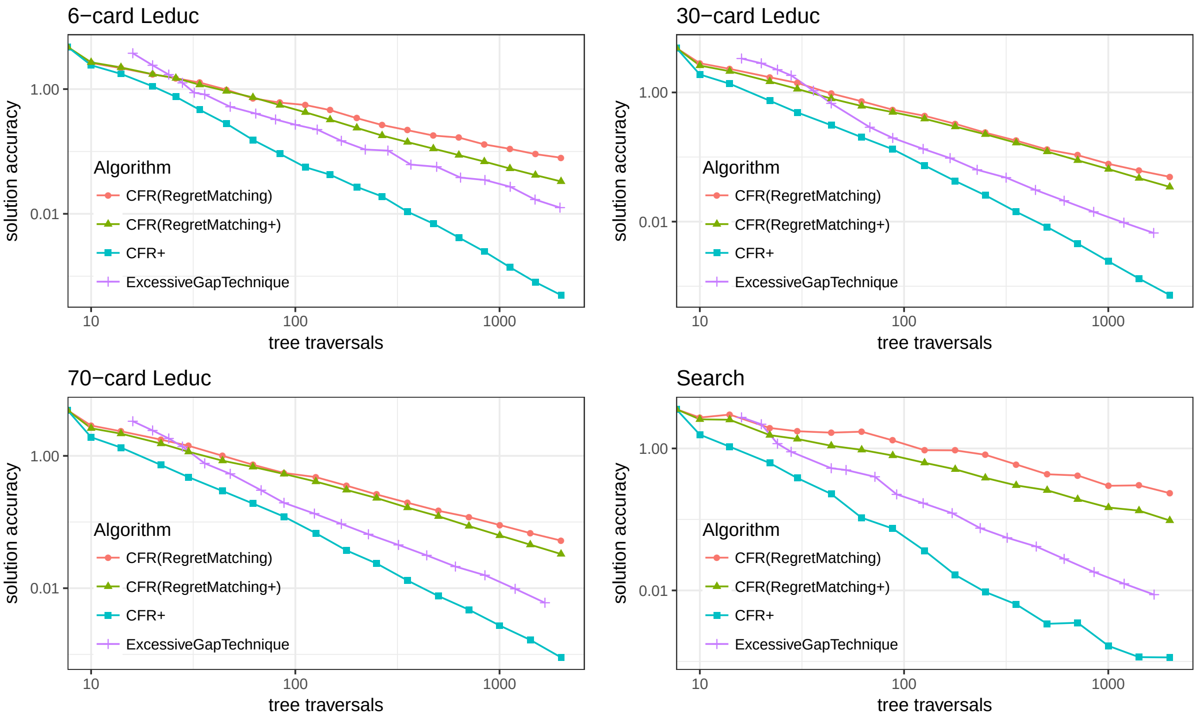

Figure 5: Solution accuracy as a function of the number of tree traversals in three different variants of Leduc hold'em and a pursuit evasion game. Results are shown for CFR with regret mathing, CFR with regret mathing$^+$, CFR$^+$, and EGT. Both axes are shown on a log scale.

in terms of theoretical convergence rate). The plots show performance on four EFGs: *Leduc poker*, a simplified poker game that is standard in EFG solving (three different deck sizes are shown), and *search*, a game played on a graph where an attacker attempts to reach a particular set of nodes, and the defender tries to capture them (full descriptions can be found in Kroer et al. [10]).

8 Stochastic Gradient Estimates

So far we have operated under the assumption that we can easily compute the matrix-vector product $g_t = Ay_t$, where A is the payoff matrix of the EFG that we are trying to solve. While g_t can indeed be computed in time linear in the size of the game tree, we may be in a case where the game tree is so large that even one traversal is too much. In that case, we are interested in developing methods that can work with some stochastic gradient estimator $\tilde{g}_t$ of the gradient. Typically, one would consider unbiased gradient estimators, i.e. $\mathbb{E}[\tilde{g}_t] = g_t$.

Assuming that we have a gradient estimator $\tilde{g}_t$ for each time t, a natural approach for attempting to compute a solution would be to apply our previous approach of running a regret minimizer for each player and using the folk theorem, but now using $\tilde{g}_t$ at each iteration, rather than g_t. If our unbiased gradient estimator $\tilde{g}_t$ is reasonably accurate then we might expect that this approach should still yield an algorithm for computing a Nash equilibrium. This turns out to be the case.

Theorem 3. *Assume that each player uses a bounded unbiased gradient estimator for their loss at* *each iteration. Then for all $p \in (0,1)$, with probability at least $1-2p$*

$$\xi(\bar{x},\bar{y}) \le \frac{\tilde{R}_T^1 + \tilde{R}_T^2}{T} + \left(2\Delta + \tilde{M}_1 + \tilde{M}_2\right)\sqrt{\frac{2}{T}\log\frac{1}{p}},$$

where $\tilde{R}_T^i$ is the regret incurred under the losses $\tilde{g}_t^i$ for player i, $\Delta = \max_{z,z' \in Z} u_2(z) - u_2(z')$ is the payoff range *of the game, and $\tilde{M}_1 \ge \max_{x,x' \in X} \langle \tilde{g}_t, x - x' \rangle, \forall \tilde{g}_t$ is a bound on the "size" of the gradient estimate, with M_2 defined analogously.*

We will not show the proof here, but it follows from introducing the discrete-time stochastic process

$$d_t := g_t(x_t - x)) - \tilde{g}_t(x_t - x),$$

observing that it is a martingale difference sequence, and applying the Azuma-Hoeffding concentration inequality.

With Theorem 3 in hand, we just need a good way to construct gradient estimates $\tilde{g}_t \approx Ay_t$. Generally, one can construct a wide array of gradient estimators by using the fact that Ay_t can be computed by traversing the EFG game tree: at each leaf node z in the tree, we add $-u_1(z)y_a$ to $g_{t,a'}$, where a is the last sequence taken by the y player, and a' is the last sequence taken by the x player. To construct an estimator, we may choose to sample actions at some subset of nodes in the game tree, and then only traverse the sampled branches, while taking care to normalize the eventual payoff so that we maintain an unbiased estimator. One of the most successful estimators construct this way is the *external sampling* estimator. In external sampling when computing the gradient Ay_t, we sample a single action at every node belonging to the y player or chance, while traversing all branches at nodes belonging to the x player.

Figure 6 shows the performance when using external sampling in CFR (CFR with sampling is usually called Monte-Carlo CFR or MCCFR), FTRL, and OMD. Performance is shown on Leduc with a 13-card deck, Goofspiel (another card game), search, and battleship. In the deterministic case we saw that CFR$^+$ was much faster than the theoretically-superior EGT algorithm (and OMD/FTRL would perform much worse than EGT). Here we see that in the stochastic case it varies which algorithm is better.

9 Historical Notes

The sequence form was discovered in the USSR in the 60s [13] and later rediscovered independently [15, 9]. Dilated DGFs for EFGs were introduced by Hoda et al. [8] where they proved that any such DGF constructed from simplex DGFs which are strongly convex must also be strongly convex. Kroer et al. [10] showed the strong convexity modulus of the dilated entropy DGF shown here. An explicit bound for the dilated Euclidean DGF can be found in Farina et al. [6], which also explores regret minimization algorithms with dilated DGFs in depth.

CFR-based algorithms were used as the algorithm for computing Nash equilibrium in all the recent milestones where AIs beat human players at various poker games [1, 12, 2, 4].

CFR was introduced by Zinkevich et al. [16]. Many later variations have been developed, for example the stochastic method MCCFR [11], and variations on which local regret minimizer to use in order to speed up practical performance [14, 3]. The proof of CFR given here is a simplified version of the more general theorem developed in [5]. The plots on CFR vs EGT are from Kroer et al. [10].

The bound on error from using a stochastic method in Theorem 3 is from Farina et al. [7], and the plots on stochastic methods are from that same paper. External sampling and several other EFG gradient estimators were introduced by Lanctot et al. [11].

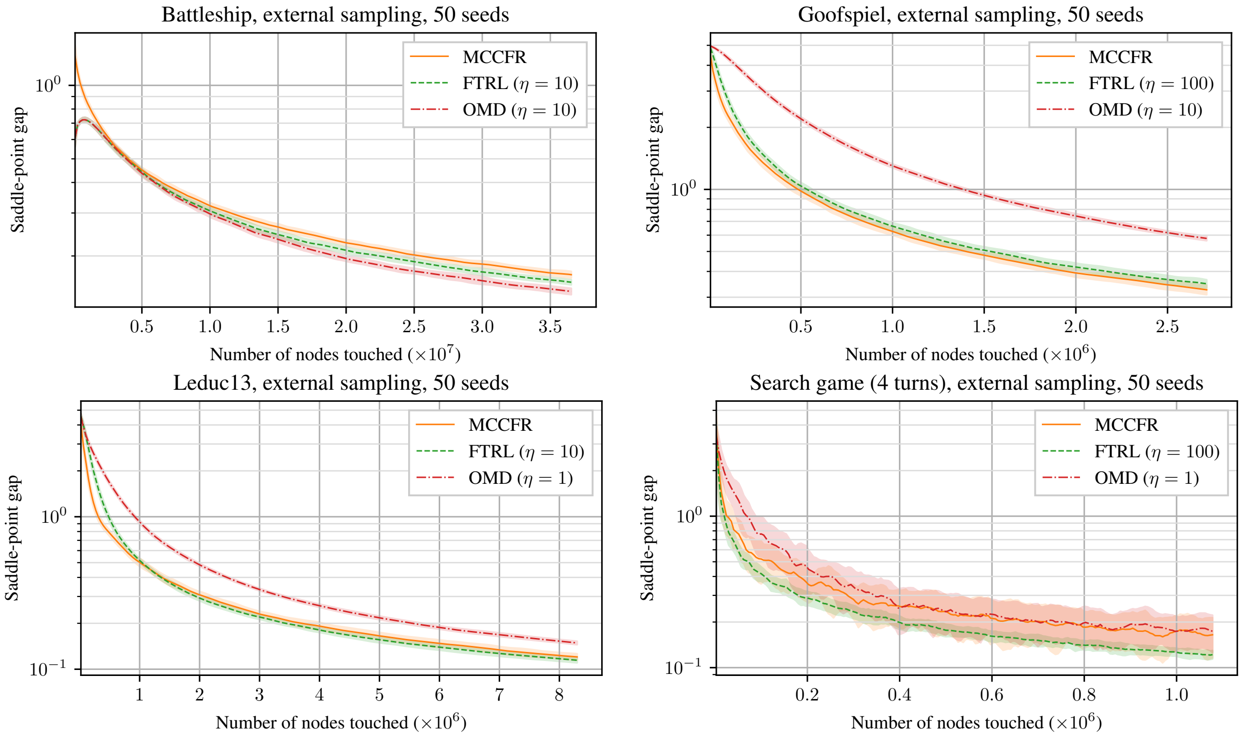

Figure 6: Performance of CFR, FTRL, and OMD when using the external sampling gradient estimator.

References

[1] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Science*, 347(6218):145–149, 2015.

[2] Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[3] Noam Brown and Tuomas Sandholm. Solving imperfect-information games via discounted regret minimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1829–1836, 2019.

[4] Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365 (6456):885–890, 2019.

[5] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Online convex optimization for sequential decision processes and extensive-form games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1917–1925, 2019.

[6] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Optimistic regret minimization for extensive-form games via dilated distance-generating functions. In *Advances in Neural Information Processing Systems*, pages 5222–5232, 2019.

[7] Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Stochastic regret minimization in extensive-form games. In *International Conference on Machine Learning*. PMLR, 2020.

[8] Samid Hoda, Andrew Gilpin, Javier Pena, and Tuomas Sandholm. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2): 494–512, 2010.

[9] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and economic behavior*, 14(2):247–259, 1996.

[10] Christian Kroer, Kevin Waugh, Fatma Kılınç-Karzan, and Tuomas Sandholm. Faster algorithms for extensive-form game solving via improved smoothing functions. *Mathematical Programming*, pages 1–33, 2020.

[11] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Advances in neural information processing systems*, pages 1078–1086, 2009.

[12] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[13] IV Romanovskii. Reduction of a game with full memory to a matrix game. *Doklady Akademii Nauk SSSR*, 144(1):62–+, 1962.

[14] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit Texas hold'em. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[15] Bernhard von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.

[16] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2007.