15-888 Computational Game Solving (Fall 2021) Tue, Oct 7th 2021

Lecture 10

Monte-Carlo CFR and offline optimization techniques

Instructor: Gabriele Farina[*]

1 Monte-Carlo CFR

In extremely large games, even performing a single iteration of CFR can be challenging. In that case, a popular technique to increase scalability is *Monte-Carlo CFR (MCCFR)* [Lanctot et al., 2009].

Remember that CFR works by orchestrating a tree of local regret minimizers (one per each decision point). At each time t, each regret minimizer outputs a local, behavioral strategy at each information set. Then, when a utility $\boldsymbol{\ell}^t$ is received as feedback by CFR, $\boldsymbol{\ell}^t$ is used to construct counterfactual utilities by considering the expected utility in each of the subtrees.

At its core, MCCFR uses the observation that if the utility vector is very sparse, then the expected utilities in each subtrees will almost always be 0, and therefore no update of the strategy is necessary for those subtrees, as no regret is cumulated.

So, the idea of MCCFR is to replace any incoming utility $\boldsymbol{\ell}^t$ by a *sparse unbiased estimator* $\tilde{\boldsymbol{\ell}}^t$, that is, a sparse vector $\tilde{\boldsymbol{\ell}}^t$ whose expectation is $\boldsymbol{\ell}^t$.

1.1 Setup and notation

Before we continue, let's recall a bit of notation about extensive-form games that will be useful for this lecture. In this lecture, it will help to think about the chance player as a real player in the game, with the caveat that its strategy is fixed and known.

We denote by $\mathcal{X}$ and $\mathcal{Y}$ the set of all sequence-form strategies for Player 1 and Player 2, respectively. We denote by $\boldsymbol{c}$ the fixed sequence-form strategy of the chance player.

For any leaf $z \in Z$, the probability that the game ends in z is the product of the probabilities of all the actions on the path from the root to z. Because of the definition of sequence-form strategies, when Player 1 and 2 play according to strategies $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{y} \in \mathcal{Y}$, respectively, this probability is equal to $\boldsymbol{x}[\sigma_1(z)] \cdot \boldsymbol{y}[\sigma_2(z)] \cdot \boldsymbol{c}[\sigma_c(z)]$, where $\sigma_i(z) \in \Sigma_i$ denotes the sequence of Player i's action on the path from the root to z. So, Player 1's expected utility is computed via the trilinear map

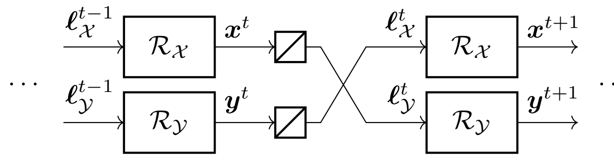
$$u_1(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) \coloneqq \sum_{z \in Z} u_1(z) \cdot \boldsymbol{x}[\sigma_1(z)] \cdot \boldsymbol{y}[\sigma_2(z)] \cdot \boldsymbol{c}[\sigma_c(z)]. \tag{1}$$

Since the strategy of the chance player is fixed, the above expression is bilinear in $\boldsymbol{x}$ and $\boldsymbol{y}$ and therefore can be expressed more concisely as $u_1(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \mathbf{A} \boldsymbol{y}$, where $\mathbf{A}$ is the usual sequence-form payoff matrix of Player 1.

[*]Computer Science Department, Carnegie Mellon University. ✉ gfarina@cs.cmu.edu.

1.2 Game-theoretic sparse unbiased utility estimator

We will illustrate how each technique can be used to construct an estimate $\tilde{\boldsymbol{\ell}}^t_{\mathcal{X}}$ for the utility $\boldsymbol{\ell}^t_{\mathcal{X}} = \mathbf{A}\boldsymbol{y}^t$ for Player 1 that is used at all times t when the players play against each other in self-play, as we have seen many times already (see also Figure 1).



$\boldsymbol{\ell}^{t-1}_{\mathcal{X}}$
$\mathcal{R}_{\mathcal{X}}$
$\boldsymbol{x}^t$
$\boldsymbol{\ell}^t_{\mathcal{X}}$
$\mathcal{R}_{\mathcal{X}}$
$\boldsymbol{x}^{t+1}$
$\cdots$
$\boldsymbol{\ell}^{t-1}_{\mathcal{Y}}$
$\mathcal{R}_{\mathcal{Y}}$
$\boldsymbol{y}^t$
$\boldsymbol{\ell}^t_{\mathcal{Y}}$
$\mathcal{R}_{\mathcal{Y}}$
$\boldsymbol{y}^{t+1}$
$\cdots$

$$\boldsymbol{\ell}^t_{\mathcal{X}} := \boldsymbol{A}\boldsymbol{y}^t$$
$$\boldsymbol{\ell}^t_{\mathcal{Y}} := -\boldsymbol{A}^\top\boldsymbol{x}^t$$

Figure 1: The flow of strategies and utilities in regret minimization for games.

The computation of an estimate for Player 2 is analogous.

External Sampling An unbiased estimator of the utility vector $\boldsymbol{\ell}^t_{\mathcal{X}} = \mathbf{A}\boldsymbol{y}^t$ can be easily constructed by independently sampling *pure* strategies $\tilde{\boldsymbol{y}}^t$ for Player 2 and $\tilde{\boldsymbol{c}}^t$ for the chance player. Indeed, as long as $\mathbb{E}_t[\tilde{\boldsymbol{y}}^t] = \boldsymbol{y}^t$ and $\mathbb{E}_t[\tilde{\boldsymbol{c}}^t] = \boldsymbol{c}$, from (1) we have that for all $\boldsymbol{x} \in \mathcal{X}$, $u_1(\boldsymbol{x}, \boldsymbol{y}^t, \boldsymbol{c}) = \mathbb{E}_t[u_1(\boldsymbol{x}, \tilde{\boldsymbol{y}}^t, \tilde{\boldsymbol{c}}^t)]$. Hence, the vector corresponding to the (random) linear function $\boldsymbol{x} \mapsto u_1(\boldsymbol{x}, \tilde{\boldsymbol{y}}^t, \tilde{\boldsymbol{c}}^t)$ is an unbiased utility estimator, called the *external sampling* utility estimator.

The external sampling utility estimator, that is, the vector corresponding to the linear function $\boldsymbol{x} \mapsto u_1(\boldsymbol{x}, \tilde{\boldsymbol{y}}^t, \tilde{\boldsymbol{c}}^t)$, can be computed via a simple traversal of the game tree. The algorithm starts at the root of the game tree and starts visiting the tree. Every time a node that belongs to the chance player or to Player 2 is encountered, an action is sampled according to the strategy $\boldsymbol{c}$ or $\boldsymbol{y}^t$, respectively. Every time a node for Player 1 is encountered, the algorithm branches on all possible actions and recurses. For every node of Player 2 or chance player, the algorithm branches on only one action. Thus computing an external sampling utility estimate is significantly cheaper to compute than the exact utility $\boldsymbol{\ell}^t_{\mathcal{X}}$.

Remark 1.1. Analogous estimators where only the chance player's strategy $\boldsymbol{c}$ or only Player 1's strategy $\boldsymbol{y}^t$ are sampled are referred to as *chance sampling* estimator and *opponent sampling* estimator, respectively.

Outcome Sampling Let $\boldsymbol{w}^t \in \mathcal{X}$ be an arbitrary strategy for Player 1. Furthermore, let $\tilde{z}^t \in Z$ be a random variable such that for all $z \in Z$,

$$\mathbb{P}_t[\tilde{z}^t = z] = \boldsymbol{w}^t[\sigma_1(z)] \cdot \boldsymbol{y}^t[\sigma_2(z)] \cdot \boldsymbol{c}[\sigma_c(z)],$$

and let $\boldsymbol{e}_z \in \mathbb{R}^{|\Sigma_1|}$ be defined as the vector such that $\boldsymbol{e}_z[\sigma_1(z)] = 1$ and $\boldsymbol{e}_z[\sigma] = 0$ for all other $\sigma \in \Sigma_1, \sigma \neq \sigma_1(z)$. It is a simple exercise to prove that the random vector

$$\tilde{\boldsymbol{\ell}}^t_{\mathcal{X}} := \frac{u_1(\tilde{z}^t)}{\boldsymbol{w}^t[\sigma_1(\tilde{z}^t)]}\boldsymbol{e}_{\tilde{z}^t}$$

is such that $\mathbb{E}_t[\tilde{\boldsymbol{\ell}}^t_{\mathcal{X}}] = \boldsymbol{\ell}^t_{\mathcal{X}}$. This particular definition of $\tilde{\boldsymbol{\ell}}^t_{\mathcal{X}}$ is called the *outcome sampling* utility estimator.

Computationally, the outcome sampling utility estimator is cheaper than the external sampling utility estimator. Indeed, since $\boldsymbol{w}^t \in \mathcal{X}$, one can sample $\tilde{z}^t$ by following a random path from the root of the game tree by sampling (from the appropriate player's strategy) one action at each node encountered along the way. The walk terminates as soon as it reaches a leaf, which corresponds to $\tilde{z}^t$.

1.3 Regret analysis

Let's analyze how much the guarantee on the regret degrades when utility estimators are used instead of exact utility vectors. To model regret minimization with unbiased utility estimators, consider the following model, which abstracts our situation.

Let $\tilde{\mathcal{R}}$ be a deterministic regret minimizer over a convex and compact set $\mathcal{X}$, and consider a second regret minimizer $\mathcal{R}$ over the same set $\mathcal{X}$ that is implemented starting from $\tilde{\mathcal{R}}$ as in Figure 2. In particular, at all times t,

- $\mathcal{R}$ queries the next decision $\boldsymbol{x}^t$ of $\tilde{\mathcal{R}}$, and outputs it;
- each utility vector $\boldsymbol{\ell}^t$ received by $\mathcal{R}$ is used by $\mathcal{R}$ to compute a *utility estimate* $\tilde{\boldsymbol{\ell}}^t$ such that

$$\mathbb{E}_t[\tilde{\boldsymbol{\ell}}^t] := \mathbb{E}[\tilde{\boldsymbol{\ell}}^t \mid \tilde{\boldsymbol{\ell}}^1, \ldots, \tilde{\boldsymbol{\ell}}^{t-1}] = \boldsymbol{\ell}^t.$$

(that is, the estimate in unbiased). The internal regret minimizer $\tilde{\mathcal{R}}$ is then shown $\tilde{\boldsymbol{\ell}}^t$ instead of $\boldsymbol{\ell}^t$.

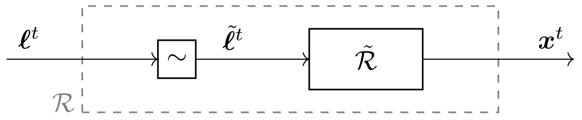


Figure 2: Abstract regret minimizer implemented by using an unbiased estimator of the utility vector at each time t.

When the estimate of the utility is very accurate, it is reasonable to expect that the regret R^T incurred by $\mathcal{R}$ up to any time T is roughly equal to the regret $\tilde{R}^T$ that is incurred by $\tilde{\mathcal{R}}$, plus some degradation term that depends on the error of the estimates. Indeed, the following can be shown [Farina et al., 2020].

Theorem 1.1. Let M and $\tilde{M}$ be constants such that

$$|(\boldsymbol{\ell}^t)^\top(\boldsymbol{x} - \boldsymbol{x}')| \le M, \quad |(\tilde{\boldsymbol{\ell}}^t)^\top(\boldsymbol{x} - \boldsymbol{x}')| \le \tilde{M}, \qquad \forall \text{ time } t, \text{ and } \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}.$$

Then, for all $\delta \in (0,1)$, with probability at least $1 - \delta$,

$$R^T \le \tilde{R}^T + (M + \tilde{M})\sqrt{2T \log \frac{1}{\delta}}.$$

Proof. Fix any $\boldsymbol{u} \in \mathcal{X}$ and introduce the discrete-time stochastic process

$$d^t := (\boldsymbol{\ell}^t)^\top(\boldsymbol{x}^t - \boldsymbol{u}) - (\tilde{\boldsymbol{\ell}}^t)^\top(\boldsymbol{x}^t - \boldsymbol{u}), \qquad t \in \mathbb{N}_{>0}.$$

Since by hypothesis $\mathbb{E}_t[\tilde{\boldsymbol{\ell}}^t] = \boldsymbol{\ell}^t$ and $\tilde{\mathcal{R}}$ is a deterministic regret minimizer, $\mathbb{E}_t[d^t] = 0$ and so $\{d^t\}$ is a martingale difference sequence. This martingale difference sequence is well-known, especially in the context of *bandit* regret minimization [Abernethy and Rakhlin, 2009, Bartlett et al., 2008].

At all times t, the maximum magnitude of d^t is given by

$$|d^t| = |(\boldsymbol{\ell}^t)^\top(\boldsymbol{z}^t - \boldsymbol{u}) - (\tilde{\boldsymbol{\ell}}^t)^\top(\boldsymbol{z}^t - \boldsymbol{u})| \le |(\boldsymbol{\ell}^t)^\top(\boldsymbol{z}^t - \boldsymbol{u})| + |(\tilde{\boldsymbol{\ell}}^t)^\top(\boldsymbol{z}^t - \boldsymbol{u})| \le M + \tilde{M}.$$

Furthermore,

$$\sum_{t=1}^T d^t = \left(\sum_{t=1}^T (\boldsymbol{\ell}^t)^\top(\boldsymbol{z}^t - \boldsymbol{u})\right) - \left(\sum_{t=1}^T (\tilde{\boldsymbol{\ell}}^t)^\top(\boldsymbol{z}^t - \boldsymbol{u})\right) = R^T(\boldsymbol{u}) - \tilde{R}^T(\boldsymbol{u}).$$

So, using the Azuma-Hoeffding concentration inequality [Hoeffding, 1963, Azuma, 1967], for all $\tau \geq 0$

$$\mathbb{P}\big[R^T(\boldsymbol{u}) \leq \tilde{R}^T(\boldsymbol{u}) + \tau\big] = 1 - \mathbb{P}\left[\sum_{t=1}^{T} d^t \geq \tau\right] \geq 1 - \exp\left\{-\frac{2\tau^2}{4T(M+\tilde{M})^2}\right\}.$$

Substituting $\tau = (M+\tilde{M})\sqrt{2T\log(1/p)}$ and taking a maximum yields the statement. □

A straightforward consequence of Theorem 1.1 is that if $\tilde{\mathcal{R}}$ guarantees sublinear regret, then also the regret in $\mathcal{R}$ will grow sublinearly with high probability.

2 Solving two-player zero-sum games via offline optimization methods

We've spent a good amount of time looking into *online* learning methods for learning strong strategies in games. In particular, we have been paying particular attention to self-play in two-player zero-sum games, where it's known that a Nash equilibrium is the solution to the bilinear saddle point problem

$$\max_{\boldsymbol{x}\in\mathcal{X}} \min_{\boldsymbol{y}\in\mathcal{Y}} \boldsymbol{x}^\top \mathbf{A}\boldsymbol{y}, \tag{2}$$

where $\mathcal{X}$ and $\mathcal{Y}$ are the sequence-form polytopes of the players, and $\mathbf{A}$ is the payoff matrix of Player 1.

In this second part of the class, we will briefly talk about what other *offline* methods are available in the literature for solving problems like (2).

2.1 Accelerated first-order methods for saddle-point optimization

The literature on convex optimization methods for bilinear saddle point problems is very rich. *Accelerated* first-order methods for such problems have been developed well before optimistic/predictive regret minimization methods (see Lecture 7) were discovered. For a long time, before RVU regret bounds and optimistic regret minimization were established, they were the only first-order methods capable of guaranteeing $O(1/T)$ convergence to a solution of (2). We briefly present two popular accelerated methods from that literature.

Excessive gap technique (EGT) The *excessive gap technique (EGT)* is a first-order method introduced by Nesterov [2005a], and one of the primary applications is to solve BSPPs such as Equation (2). EGT assumes access to a proximal setup for $\mathcal{X}$ and $\mathcal{Y}$, with one-strongly-convex (also known as *distance-generating functions (DGFs)*) d_x, d_y, and constructs smoothed approximations of the optimization problems faced by the x and y players. Based on this setup, we formally state the EGT of Nesterov [2005b] in Algorithm 1. EGT alternatingly takes steps focused on decreasing one or the other smoothing parameter. These steps are called ShrinkX and ShrinkY in Algorithm 1, and make use of some of the concepts related to distance-generating functions that we have seen in Lecture 7.

Algorithm 1 shows how initial points are selected and the alternating steps and stepsizes are computed. Nesterov [2005b] proves that the EGT algorithm converges at a rate of $O(1/T)$.

Theorem 2.1 (Nesterov [2005b], Theorem 6.3)**.** At every iteration $t \geq 1$ of the EGT algorithm, the iterate $(\boldsymbol{x}^t, \boldsymbol{y}^t)$ produced by EGT (Algorithm 1) satisfies $\boldsymbol{x}^t \in \mathcal{X}$, $\boldsymbol{y}^t \in \mathcal{Y}$, and

$$\gamma(\boldsymbol{x}^t, \boldsymbol{y}^t) \coloneqq \max_{\boldsymbol{y}\in\mathcal{Y}} (\boldsymbol{x}^t)^\top \mathbf{A}\boldsymbol{y} - \min_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{x}^\top \mathbf{A}\boldsymbol{y}^t \leq \frac{4\|\mathbf{A}\|\sqrt{\Omega_{d_x,\mathcal{X}}\Omega_{d_y,\mathcal{Y}}}}{t+1}.$$

Algorithm 1: Excessive Gap Technique (EGT) algorithm.

```
function INITIALIZE()
    $t \leftarrow 0$
    $\mu_x^0 \leftarrow \|\mathbf{A}\|, \quad \mu_y^0 \leftarrow \|\mathbf{A}\|$
    $\tilde{\boldsymbol{x}} \leftarrow \arg\min_{\hat{x}\in\mathcal{X}} d_x(\hat{x})$
    $\boldsymbol{y}^0 \leftarrow \nabla d_y^*(\mathbf{A}^\top \tilde{\boldsymbol{x}}/\mu_y^0)$
    $\boldsymbol{x}^0 \leftarrow \text{prox}_{\tilde{\boldsymbol{x}}}(\mathbf{A}\boldsymbol{y}^0/\mu_x^0)$

function ITERATE()
    $t \leftarrow t+1, \quad \tau \leftarrow 2/(t+2)$
    if $t$ is even then SHRINKX()
    else SHRINKY()

function SHRINKX()
    $\bar{\boldsymbol{x}} \leftarrow -\nabla d_x^*(-\mathbf{A}\boldsymbol{y}^{t-1}/\mu_x^{t-1})$
    $\hat{\boldsymbol{x}} \leftarrow (1-\tau)\boldsymbol{x}^{t-1} + \tau\bar{\boldsymbol{x}}$
    $\bar{\boldsymbol{y}} \leftarrow \nabla d_y^*(\mathbf{A}^\top \hat{\boldsymbol{x}}/\mu_y^{t-1})$
    $\tilde{\boldsymbol{x}} \leftarrow \text{prox}_{\bar{\boldsymbol{x}}}\left(\frac{\tau}{(1-\tau)\mu_x^{t-1}}\mathbf{A}\bar{\boldsymbol{y}}\right)$
    $\boldsymbol{x}^t \leftarrow (1-\tau)\boldsymbol{x}^{t-1} + \tau\tilde{\boldsymbol{x}}$
    $\boldsymbol{y}^t \leftarrow (1-\tau)\boldsymbol{y}^{t-1} + \tau\bar{\boldsymbol{y}}$
    $\mu_x^t \leftarrow (1-\tau)\mu_x^{t-1}$

function SHRINKY()
    $\bar{\boldsymbol{y}} \leftarrow \nabla d_y^*(\mathbf{A}^\top \boldsymbol{x}^{t-1}/\mu_y^{t-1})$
    $\hat{\boldsymbol{y}} \leftarrow (1-\tau)\boldsymbol{y}^{t-1} + \tau\bar{\boldsymbol{y}}$
    $\bar{\boldsymbol{x}} \leftarrow -\nabla d_x^*(-\mathbf{A}\hat{\boldsymbol{y}}/\mu_x^{t-1})$
    $\tilde{\boldsymbol{y}} \leftarrow \text{prox}_{\bar{\boldsymbol{y}}}\left(\frac{-\tau}{(1-\tau)\mu_y^{t-1}}\mathbf{A}^\top\bar{\boldsymbol{x}}\right)$
    $\boldsymbol{y}^t \leftarrow (1-\tau)\boldsymbol{y}^{t-1} + \tau\tilde{\boldsymbol{y}}$
    $\boldsymbol{x}^t \leftarrow (1-\tau)\boldsymbol{x}^{t-1} + \tau\bar{\boldsymbol{x}}$
    $\mu_y^t \leftarrow (1-\tau)\mu_y^{t-1}$
```

Mirror prox (MP) Rather than construct smoothed approximations, the *Mirror Prox (MP)* algorithm Nemirovski [2004] directly uses the DGFs to take first-order steps. Hence, the MP algorithm is best understood as an algorithm that operates on the product space $\mathcal{X}\times\mathcal{Y}$ directly. As such, in most analyses of the MP algorithm, a single 1-strongly convex DGF for the product space $\mathcal{X}\times\mathcal{Y}$ is required. To better align with the setup used for EGT, we will define the DGF for the product space $\mathcal{X}\times\mathcal{Y}$ starting from proximal setups for both $\mathcal{X}$ and $\mathcal{Y}$, with 1-strongly convex DGFs d_x, d_y with respect to norms $\|\cdot\|_x$ and $\|\cdot\|_y$, respectively. Algorithm 2 shows the sequence of steps taken in every iteration of the MP algorithm. Compared to EGT, mirror prox has a somewhat simpler structure: it simply takes repeated extrapolated proximal steps. First, a proximal step in the descent direction is taken for both x and y. Then, the utility at those new points is used to take a proximal step starting from the previous iterate (this is the extrapolation part: a step is taken starting from the previous iterate, but with the extrapolated utility). Finally, the *average* strategy is output.

Algorithm 2: Mirror Prox (MP) algorithm.

```
function INITIALIZE()
    $t \leftarrow 0$
    $\boldsymbol{z}_x^0 \leftarrow \arg\min_{\hat{\boldsymbol{x}}\in\mathcal{X}} d_x(\hat{\boldsymbol{x}})$
    $\boldsymbol{z}_y^0 \leftarrow \arg\min_{\hat{\boldsymbol{y}}\in\mathcal{Y}} d_y(\hat{\boldsymbol{y}})$

function ITERATE()
    $t \leftarrow t+1$
    $\boldsymbol{w}_x^t \leftarrow \text{prox}_{\boldsymbol{z}_x^t}\left(\eta^t \mathbf{A}\boldsymbol{z}_y^{t-1}\right)$
    $\boldsymbol{w}_y^t \leftarrow \text{prox}_{\boldsymbol{z}_y^t}\left(-\eta^t \mathbf{A}^\top \boldsymbol{z}_x^{t-1}\right)$
    $\boldsymbol{z}_x^{t+1} \leftarrow \text{prox}_{\boldsymbol{z}_x^t}\left(\eta^t \mathbf{A}\boldsymbol{w}_y^t\right)$
    $\boldsymbol{z}_y^{t+1} \leftarrow \text{prox}_{\boldsymbol{z}_y^t}\left(-\eta^t \mathbf{A}^\top \boldsymbol{w}_x^t\right)$
    $\boldsymbol{x}^t \leftarrow [\sum_{\tau=1}^t \eta^\tau]^{-1} \sum_{\tau=1}^t \eta^\tau \boldsymbol{w}_x^\tau$
    $\boldsymbol{y}^t \leftarrow [\sum_{\tau=1}^t \eta^\tau]^{-1} \sum_{\tau=1}^t \eta^\tau \boldsymbol{w}_y^\tau$
```

Note: $\{\eta^t\}$ is a sequence of stepsize parameters. A well-known and theoretically-sound choice for η^t is $\eta^t := \frac{1}{\|A\|}$ for all $t = 0, 1, \dots$ (see also Theorem 2.2).

As we recall in the next theorem, like EGT the mirror prox algorithm converges at rate $O(1/T)$.

Theorem 2.2 (Ben-Tal and Nemirovski [2001], Theorem 5.5.1)**.** Suppose the stepsize in Algorithm 2 is set as $\eta_t = 1/\|A\|$. Then we have

$$\gamma(\boldsymbol{x}^t, \boldsymbol{y}^t) := \max_{\boldsymbol{y}\in\mathcal{Y}} (\boldsymbol{x}^t)^\top \mathbf{A}\boldsymbol{y} - \min_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{x}^\top \mathbf{A}\boldsymbol{y}^t \le \frac{\|\mathbf{A}\|(\Omega_{d_x,\mathcal{X}} + \Omega_{d_y,\mathcal{Y}})}{2t}.$$

2.2 Linear programming formulation

Every bilinear saddle-point problem where $\mathcal{X}$ and $\mathcal{Y}$ are polytopes can always be converted into a linear program. The technique is completely general and based on linear programming duality.

Define the function

$$g(\boldsymbol{x}) \coloneqq \min_{\boldsymbol{y}\in\mathcal{Y}} \boldsymbol{x}^\top \mathbf{A}\boldsymbol{y},$$

which appears as the internal minimization problem in (2), so that

$$\max_{\boldsymbol{x}\in\mathcal{X}} \min_{\boldsymbol{y}\in\mathcal{Y}} \boldsymbol{x}^\top \mathbf{A}\boldsymbol{y} = \max_{\boldsymbol{x}\in\mathcal{X}} g(\boldsymbol{x}). \tag{3}$$

Remember from Lecture 2 that the sequence-form polytope $\mathcal{Y}$ of Player 2 that appears in the domain of the minimization is defined as the intersection of the linear equality and inequality constraints

$$\mathcal{Y} \coloneqq \left\{ \boldsymbol{y} \in \mathbb{R}^{|\Sigma_2|}_{\geq 0} : \quad \sum_{a\in A_j} \boldsymbol{y}[ja] = \begin{cases} 1 & \text{if } p_j = \varnothing \\ \boldsymbol{y}[p_j] & \text{otherwise} \end{cases} \quad \forall j \in \mathcal{J}_2 \right\}.$$

We can represent the linear equality and inequality constraints that define $\mathcal{Y}$ more compactly in vector form as $\mathcal{Y} = \{\boldsymbol{y} : \mathbf{F}_2\boldsymbol{y} = \boldsymbol{f}_2, \boldsymbol{y} \geq \mathbf{0}\}$ for an appropriate matrix $\mathbf{F}_2$ and vector $\boldsymbol{f}_2$. (Similarly, for $\mathcal{X}$ we have $\mathcal{X} = \{\boldsymbol{x} : \mathbf{F}_1\boldsymbol{x} = \boldsymbol{f}_1, \boldsymbol{x} \geq \mathbf{0}\}$.) With that, we can rewrite $g(\boldsymbol{x})$ as the linear program

$$g(\boldsymbol{x}) = \begin{cases} \min & (\mathbf{A}^\top\boldsymbol{x})^\top\boldsymbol{y} \\ \text{s.t.} & \text{①}\ \mathbf{F}_2\boldsymbol{y} = \boldsymbol{f}_2 \\ & \text{②}\ \boldsymbol{y} \geq \mathbf{0}. \end{cases} \tag{P}$$

From the duality theory of linear programs, we can rewrite the linear program (P) as

$$g(\boldsymbol{x}) = \begin{cases} \max & \boldsymbol{f}_2^\top\boldsymbol{v} \\ \text{s.t.} & \text{①}\ \mathbf{F}_2^\top\boldsymbol{v} \leq \mathbf{A}^\top\boldsymbol{x} \\ & \text{②}\ \boldsymbol{v} \text{ free.} \end{cases} \tag{D}$$

By plugging the above expression for $g(\boldsymbol{x})$ back into (3), and expanding the definition of $\mathcal{X} = \{\boldsymbol{x} : \mathbf{F}_1\boldsymbol{x} = \boldsymbol{f}_1, \boldsymbol{x} \geq \mathbf{0}\}$, we obtain the following characterization of Nash equilibrium strategies in two-player zero-sum games based on linear programming.

Proposition 2.1. Let $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^{|\Sigma_1|} : \mathbf{F}_1\boldsymbol{x} = \boldsymbol{f}_1, \boldsymbol{x} \geq \mathbf{0}\}$ and $\mathcal{Y} = \{\boldsymbol{y} \in \mathbb{R}^{|\Sigma_2|} : \mathbf{F}_2\boldsymbol{y} = \boldsymbol{f}_2, \boldsymbol{y} \geq \mathbf{0}\}$ be the sequence-form polytopes of the two players, and let $\mathbf{A}$ be the payoff matrix for Player 1. Then, a Nash equilibrium strategy for Player 1 can be computed as the solution $\boldsymbol{x}$ to the linear program

$$\begin{cases} \max & \boldsymbol{f}_2^\top\boldsymbol{v} \\ \text{s.t.} & \text{①}\ \mathbf{A}^\top\boldsymbol{x} - \mathbf{F}_2^\top\boldsymbol{v} \geq \mathbf{0} \\ & \text{②}\ \mathbf{F}_1\boldsymbol{x} = \boldsymbol{f}_1 \\ & \text{③}\ \boldsymbol{x} \geq \mathbf{0},\ \ \boldsymbol{v} \text{ free.} \end{cases} \tag{4}$$

The linear program for a Nash equilibrium strategy of Player 2 can be obtained by swapping the roles of the two players, and taking $\mathbf{A}$ to be the payoff matrix of Player 2 instead, which is simply the opposite of the transpose of the payoff matrix of Player 1 in a zero-sum game. Specifically, with the same notation as Proposition 2.1, a Nash equilibrium strategy for Player 2 is the solution $\boldsymbol{y}$ to the linear program

$$\begin{cases} \max & \boldsymbol{f}_1^\top\boldsymbol{v} \\ \text{s.t.} & \text{①}\ -\mathbf{A}\boldsymbol{y} - \mathbf{F}_1^\top\boldsymbol{v} \geq \mathbf{0} \\ & \text{②}\ \mathbf{F}_2\boldsymbol{y} = \boldsymbol{f}_2 \\ & \text{③}\ \boldsymbol{y} \geq \mathbf{0},\ \ \boldsymbol{v} \text{ free.} \end{cases}$$

Payoff matrix sparsification Generally speaking, the more nonzeros in the linear program, the longer it will take solvers to solve the linear program. In the particular case of the Nash equilibrium strategy linear program given in (4), the number of nonzeros in the constraint matrix of the linear program is mostly driven by the number of nonzeros in the payoff matrix $\mathbf{A}$ of the game.

To further push the limit as to what size games linear programming solvers can be used on, Zhang and Sandholm [2020] made the following observation. Suppose that the payoff matrix $\mathbf{A}$ of the game can be expressed in the form

$$\mathbf{A} = \hat{\mathbf{A}} + \mathbf{U}\mathbf{M}^{-1}\mathbf{V}^\top, \tag{5}$$

for some matrices $\hat{\mathbf{A}}$, $\mathbf{U}$, $\mathbf{M}^{-1}$ (where $\mathbf{M}$ is an invertible matrix), and $\mathbf{V}$. We call expression (5) a *sparsification* of $\mathbf{A}$. Then, given the sparsication, we have that the term $\mathbf{A}^\top \boldsymbol{x}$ that appears in constraint ② of (4) as

$$\mathbf{A}^\top \boldsymbol{x} = \hat{\mathbf{A}}^\top \boldsymbol{x} + \mathbf{V}\mathbf{M}^{-\top}\mathbf{U}^\top \boldsymbol{x}.$$

So, by introducing the vector $\boldsymbol{w} := \mathbf{M}^{-\top}\mathbf{U}^\top \boldsymbol{x}$, we can write

$$\mathbf{A}^\top \boldsymbol{x} = \hat{\mathbf{A}}^\top \boldsymbol{x} + \mathbf{V}\boldsymbol{w}.$$

Given that $\boldsymbol{w} = \mathbf{M}^{-\top}\mathbf{U}^\top \boldsymbol{x} \iff \mathbf{M}^\top \boldsymbol{w} = \mathbf{U}^\top \boldsymbol{x}$, we can therefore rewrite the linear program (4) into its sparsified form as (6).

$$\begin{cases} \max & \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{s.t.} & ① \ \mathbf{A}^\top \boldsymbol{x} - \mathbf{F}_2^\top \boldsymbol{v} \ge \mathbf{0} \\ & ② \ \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ & ③ \ \boldsymbol{x} \ge \mathbf{0}, \ \ \boldsymbol{v} \text{ free} \end{cases} \xrightarrow{\text{sparsified}} \begin{cases} \max & \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{s.t.} & ① \ \hat{\mathbf{A}}^\top \boldsymbol{x} - \mathbf{F}_2^\top \boldsymbol{v} + \mathbf{V}\boldsymbol{w} \ge \mathbf{0} \\ & ② \ \mathbf{M}^\top \boldsymbol{w} - \mathbf{U}^\top \boldsymbol{x} = \mathbf{0} \\ & ③ \ \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ & ④ \ \boldsymbol{x} \ge \mathbf{0}, \ \ \boldsymbol{v} \text{ free}, \ \ \boldsymbol{w} \text{ free}. \end{cases} \tag{6}$$

References

Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2009.

Gabriele Farina, Christian Kroer, and Tuomas Sandholm. Stochastic regret minimization in extensive-form games. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Jacob Abernethy and Alexander Rakhlin. Beating the adaptive bandit with high probability. Technical Report UCB/EECS-2009-10, EECS Department, University of California, Berkeley, 2009. URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-10.html.

Peter L. Bartlett, Varsha Dani, Thomas Hayes, Sham Kakade, Alexander Rakhlin, and Ambuj Tewari. High-probability regret bounds for bandit online linear optimization. In *Proceedings of the Conference on Learning Theory (COLT)*, 2008.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal*, 19 (3), 1967.

Yurii Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103, 2005a.

Yurii Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal of Optimization*, 16(1), 2005b.

Arkadi Nemirovski. Prox-method with rate of convergence O(1/t) for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1), 2004.

Ahron Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*, volume 2. Siam, 2001.

Brian Hu Zhang and Tuomas Sandholm. Sparsified linear programming for zero-sum equilibrium finding. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.