15-888 Computational Game Solving (Fall 2025) Released on: Sep. 28, 2025

Homework 1

Student: boxiangf **Due on: Oct. 10, 2025**

Instructions Submit your homework on Gradescope. Submit a single `pdf` file containing **both** your written solutions **and** your code (`stub.py`) attached to the end (for example, using a `verbatim` environment). You can (but are not required to) typeset your written solutions in the `.tex` file provided in the homework `.zip`.

1 A Regret-based proof of the minimax theorem (20 points)

Let $\mathcal{X} \subseteq \mathbb{R}^{m_1}$ and $\mathcal{Y} \subseteq \mathbb{R}^{m_2}$ be convex and compact sets, and $\mathbf{A} \in \mathbb{R}^{m_1 \times m_2}$. The minimax theorem asserts that

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle = \max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle.$$

In this problem, you will show that the mere existence of (external) regret minimization algorithms is powerful enough to imply the minimax theorem.

One direction (called *weak duality*) of the proof is easy and very general. Specifically, show the following.

Problem 1.1 (5 points)**.** Show that

$$\min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle \geq \max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle. \tag{1}$$

Solution. Define $f(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle$. By definition of the minimum and maximum, for all $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{y} \in \mathcal{Y}$, we have

$$\min_{\boldsymbol{x'} \in \mathcal{X}} f(\boldsymbol{x'}, \boldsymbol{y}) \leq f(\boldsymbol{x}, \boldsymbol{y}) \leq \max_{\boldsymbol{y'} \in \mathcal{Y}} f(\boldsymbol{x}, \boldsymbol{y'}).$$

Define

$$g(\boldsymbol{y}) \coloneqq \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}, \boldsymbol{y}) \qquad h(\boldsymbol{x}) \coloneqq \max_{\boldsymbol{y} \in \mathcal{Y}} f(\boldsymbol{x}, \boldsymbol{y})$$

so for all $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{y} \in \mathcal{Y}$, we have $g(\boldsymbol{y}) \leq h(\boldsymbol{x})$. So the maximum of $g(\boldsymbol{y})$ is smaller or equal to the minimum of $h(\boldsymbol{x})$:

$$\max_{\boldsymbol{y} \in \mathcal{Y}} g(\boldsymbol{y}) \leq \min_{\boldsymbol{x} \in \mathcal{X}} h(\boldsymbol{x}).$$

Recalling the definitions of f, g, and h, we obtain

$$\max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle \leq \min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle.$$

□

To show the reverse inequality, we will employ regret minimization to solve the bilinear saddle point problem $\max_{\boldsymbol{y}\in\mathcal{Y}} \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle$. At each time t, we will let a regret minimizer $\mathfrak{R}_{\mathcal{Y}}$ pick strategies $\boldsymbol{y}^{(t)} \in \mathcal{Y}$, whereas we will always assume that $\boldsymbol{x}^{(t)} \in \mathcal{X}$ is chosen by the environment to best respond to $\boldsymbol{y}^{(t)}$, that is,

$$\boldsymbol{x}^{(t)} \in \arg\min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)}\rangle.$$

The utility function observed by $\mathfrak{R}_{\mathcal{Y}}$ at each time t is set to

$$u_{\mathcal{Y}}^t : \boldsymbol{y} \mapsto \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}\rangle = \langle \boldsymbol{y}, \mathbf{A}^\top \boldsymbol{x}^{(t)}\rangle.$$

We will assume that $\mathfrak{R}_{\mathcal{Y}}$ guarantees *sublinear regret* under any sequence of utilities; we have seen many algorithms with this property, such as `FTRL` and `RM`.

Let $\bar{\boldsymbol{x}}^{(T)} \in \mathcal{X}$ and $\bar{\boldsymbol{y}}^{(T)} \in \mathcal{Y}$ be the average strategies output up to time T, that is,

$$\bar{\boldsymbol{x}}^{(T)} := \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}^{(t)} \qquad \bar{\boldsymbol{y}}^{(T)} := \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{y}^{(t)}.$$

Problem 1.2 (5 points)**.** Argue that at all t,

$$\max_{\boldsymbol{y}\in\mathcal{Y}} \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \geq \frac{1}{T} \min_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{T} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)}\rangle \geq \frac{1}{T}\sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}^{(t)}\rangle. \tag{2}$$

★ Hint: use the definition of $\bar{\boldsymbol{y}}^{(T)}$. Then, use the information you have on $\boldsymbol{x}^{(t)}$.

Solution. We define $g(\boldsymbol{y}) := \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle$. Since $\bar{\boldsymbol{y}}^{(T)} \in \mathcal{Y}$, we have

$$\max_{\boldsymbol{y}\in\mathcal{Y}} g(\boldsymbol{y}) \geq g(\bar{\boldsymbol{y}}^{(T)}).$$

Recalling the definition of g, we have

$$\max_{\boldsymbol{y}\in\mathcal{Y}} \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \geq \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\bar{\boldsymbol{y}}^{(T)}\rangle. \tag{3}$$

Recall the definition $\bar{\boldsymbol{y}}^{(T)} := \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{y}^{(t)}$. Using the linearity of the inner product operator for the second argument (in real vector spaces), we have

$$\min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\bar{\boldsymbol{y}}^{(T)}\rangle = \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\cdot\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{y}^{(t)}\rangle = \frac{1}{T}\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{T} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)}\rangle. \tag{4}$$

Combining (3) and (4), we obtain

$$\max_{\boldsymbol{y}\in\mathcal{Y}} \min_{\boldsymbol{x}\in\mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \geq \frac{1}{T}\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{T} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)}\rangle. \tag{5}$$

By the super-additivity property of the minimum, for any function $f_t(\boldsymbol{x})$, we have

$$\min_{\boldsymbol{x}\in\mathcal{X}} \sum_{t=1}^{T} f_t(\boldsymbol{x}) \geq \sum_{t=1}^{T} \min_{\boldsymbol{x}\in\mathcal{X}} f_t(\boldsymbol{x}).$$

With $f_t(\boldsymbol{x}) \coloneqq \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle$, we have

$$\frac{1}{T} \min_{\boldsymbol{x} \in \mathcal{X}} \sum_{t=1}^{T} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle \geq \frac{1}{T} \sum_{t=1}^{T} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle. \tag{6}$$

Recall the definition of $\boldsymbol{x}^{(t)} \in \arg\min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle$. This means that

$$\min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle = \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle. \tag{7}$$

Combining (5), (6), and (7), we obtain

$$\max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle \geq \frac{1}{T} \min_{\boldsymbol{x} \in \mathcal{X}} \sum_{t=1}^{T} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle \geq \frac{1}{T} \sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle.$$

□

Problem 1.3 (5 points). Let $\mathsf{Reg}_{\mathcal{Y}}^{(T)}$ be the regret accumulated by $\mathfrak{R}_{\mathcal{Y}}$ up to time T. Using (2), argue that at all times T

$$\max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle \geq \min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle - \frac{\mathsf{Reg}_{\mathcal{Y}}^{(T)}}{T}. \tag{8}$$

★ Hint: use the definition of regret $\mathsf{Reg}_{\mathcal{Y}}^{(T)}$.

Solution. From (2), we have

$$\max_{\boldsymbol{y} \in \mathcal{Y}} \min_{\boldsymbol{x} \in \mathcal{X}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle \geq \frac{1}{T} \sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle.$$

Recall the definition of regret:

$$\mathsf{Reg}_{\mathcal{Y}}^{(T)} = \max_{\boldsymbol{y} \in \mathcal{Y}} \sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y} \rangle - \sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle.$$

Multiplying by $\frac{1}{T}$ and rearranging, we have

$$\frac{1}{T} \sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}^{(t)} \rangle = \frac{1}{T} \max_{\boldsymbol{y} \in \mathcal{Y}} \sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y} \rangle - \frac{\mathsf{Reg}_{\mathcal{Y}}^{(T)}}{T}. \tag{9}$$

Noting the definition of the average strategy and the linearity of the inner product operator for the first argument, we have

$$\frac{1}{T} \max_{\boldsymbol{y} \in \mathcal{Y}} \sum_{t=1}^{T} \langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y} \rangle = \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \bar{\boldsymbol{x}}^{(T)}, \mathbf{A}\boldsymbol{y} \rangle \tag{10}$$

with $\bar{\boldsymbol{x}}^{(T)} \in \mathcal{X}$. So

$$\max_{\boldsymbol{y} \in \mathcal{Y}} \langle \bar{\boldsymbol{x}}^{(T)}, \mathbf{A}\boldsymbol{y} \rangle \geq \min_{\boldsymbol{x} \in \mathcal{X}} \max_{\boldsymbol{y} \in \mathcal{Y}} \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle \tag{11}$$

with the quantity on the right-hand-side being the minimum over $\mathcal{X}$. Combining (9), (10), and (11), we have

$$\frac{1}{T}\sum_{t=1}^{T}\langle \boldsymbol{x}^{(t)}, \mathbf{A}\boldsymbol{y}^{(t)}\rangle \geq \min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle - \frac{\mathsf{Reg}_{\mathcal{Y}}^{(T)}}{T}.$$

Comparing to (2), we obtain

$$\max_{\boldsymbol{y}\in\mathcal{Y}}\min_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \geq \min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle - \frac{\mathsf{Reg}_{\mathcal{Y}}^{(T)}}{T}.$$

□

Problem 1.4 (5 points). Use (8) and (1) and to conclude that

$$\max_{\boldsymbol{y}\in\mathcal{Y}}\min_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle = \min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle,$$

Solution. From (1), we have

$$\max_{\boldsymbol{y}\in\mathcal{Y}}\min_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \leq \min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle.$$

From (8), we have for all times T

$$\max_{\boldsymbol{y}\in\mathcal{Y}}\min_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \geq \min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle - \frac{\mathsf{Reg}_{\mathcal{Y}}^{(T)}}{T}.$$

Combining both, we have

$$\min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle - \frac{\mathsf{Reg}_{\mathcal{Y}}^{(T)}}{T} \leq \max_{\boldsymbol{y}\in\mathcal{Y}}\min_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \leq \min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle.$$

Let $T \to \infty$. Since the regret is sublinear, we have $\lim_{T\to\infty}\frac{\mathsf{Reg}_{\mathcal{Y}}^{(T)}}{T} = 0$. So letting $T \to \infty$ forces the inequality

$$\min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \leq \max_{\boldsymbol{y}\in\mathcal{Y}}\min_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle \leq \min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle$$

and thus

$$\max_{\boldsymbol{y}\in\mathcal{Y}}\min_{\boldsymbol{x}\in\mathcal{X}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle = \min_{\boldsymbol{x}\in\mathcal{X}}\max_{\boldsymbol{y}\in\mathcal{Y}}\langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y}\rangle.$$

□

2 Deriving CFR from regret circuits

In this problem, you will derive CFR using regret circuits. For your convenience, we include the pseudocode from Lecture 5 for the subroutine that computes the counterfactual utilities (Algorithm 1).

Recall that the sequence-form strategy set can be decomposed as follows. First, in any observation point $k \in \mathcal{K}$ with a set of children $\{p_1, \ldots, p_\nu\} = \{\rho(k, s) : s \in \mathcal{S}_k\}$,

$$\mathcal{X}_k = \mathcal{X}_{p_1} \times \mathcal{X}_{p_2} \times \cdots \times \mathcal{X}_{p_\nu}, \tag{12}$$

where $\mathcal{X}_{p_i}$ is the sequence-form polytope corresponding to the subtree rooted at point p_i. Second, in any decision point $j \in \mathcal{J}$ with a set of children $\{p_1, \dots, p_\nu\} = \{\rho(j,a) : a \in \mathcal{A}_j\}$,

$$\mathcal{X}_j \coloneqq \text{co}\left\{ \begin{pmatrix} \boldsymbol{e}_1 \\ \mathcal{X}_{p_1} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix}, \begin{pmatrix} \boldsymbol{e}_2 \\ \mathbf{0} \\ \mathcal{X}_{p_2} \\ \vdots \\ \mathbf{0} \end{pmatrix}, \dots, \begin{pmatrix} \boldsymbol{e}_\nu \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathcal{X}_{p_\nu} \end{pmatrix} \right\}, \tag{13}$$

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

Algorithm 1: Counterfactual regret minimization (`CFR`)

Input: A regret minimizer $\mathfrak{R}_j$ for each decision point $j \in \mathcal{J}$ of the tree-form decision process.

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 $j = v$;
      $V^{(t)}[j] \coloneqq \sum_{a \in \mathcal{A}_j} b_j^{(t)}[a] \cdot (\boldsymbol{u}^{(t)}[(j,a)] + V^{(t)}[\rho(j,a)])$;
    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($\boldsymbol{u}_j^{(t)}$);

You will prove soundness of `CFR` by induction, proceeding in a bottom-up fashion.

Problem 2.1 (5 points). Argue that the utility at every terminal decision point $j \in \mathcal{J}$ is given by $\boldsymbol{u}_j^{(t)}[a] = \boldsymbol{u}^{(t)}[(j,a)]$ for all $a \in \mathcal{A}_j$.

Solution. At a decision node $j \in \mathcal{J}$, the utility of $\boldsymbol{x}_j \in \mathcal{X}_j$ is given as a convex combination of action probabilities over the columns of $\mathcal{X}_j$. Referring to Equation 13, the utility for action $a \in \mathcal{A}_j$ is given by

$$\boldsymbol{u}_j^{(t)}[a] = \langle (\boldsymbol{e}_a, \mathbf{0}, \cdots, \boldsymbol{x}_{\rho(j,a)}, \cdots, \mathbf{0}), \boldsymbol{u}_j^{(t)} \rangle.$$

Expanding the inner product and noting that the node is terminal, we obtain

$$\boldsymbol{u}_j^{(t)}[a] = \langle \boldsymbol{e}_a, \boldsymbol{u}_j^{(t)} \rangle + \langle \boldsymbol{x}_\perp, \boldsymbol{u}_j^{(t)} \rangle$$

Note a abuse of notation in that $\boldsymbol{u}_j^{(t)}$ is the matching block to the LHS vector. The first inner product is exactly the definition of $\boldsymbol{u}^{(t)}[(j,a)]$ while the second inner product evaluates to zero since $\boldsymbol{x}_\perp = \mathbf{0}$. Thus,

$$\boldsymbol{u}_j^{(t)}[a] = \langle \boldsymbol{e}_a, \boldsymbol{u}_j^{(t)} \rangle + \langle \mathbf{0}, \boldsymbol{u}_j^{(t)} \rangle = \boldsymbol{u}^{(t)}[(j,a)] + 0 = \boldsymbol{u}^{(t)}[(j,a)]$$

for all $a \in \mathcal{A}_j$. □

Problem 2.2 (5 points). Apply the regret circuit for the Cartesian product in each observation node $p \in \mathcal{K}$. Derive the utility vectors given as input to the children of p.

★ Hint: Use the decomposition given in (12).

Solution. At each observation node $p \in \mathcal{K}$, the regret circuit obtains a utility vector $\boldsymbol{u}_p^{(t)} = (\boldsymbol{u}_{p_i}^{(t)})_{p_i \in C_p}$, where C_p is the set of children of observation p. Therefore, $\boldsymbol{u}_p^{(t)}$ can be decomposed as

$$\boldsymbol{u}_p^{(t)} = \begin{pmatrix} \boldsymbol{u}_{p_1}^{(t)} \\ \boldsymbol{u}_{p_2}^{(t)} \\ \vdots \\ \boldsymbol{u}_{p_\nu}^{(t)} \end{pmatrix} \tag{14}$$

where p_i corresponds to the i'th children of node p. By the regret circuit algorithm on a Cartesian product, it would forward the utility vector $\boldsymbol{u}_{p_i}^{(t)}$ to child $p_i \in C_p$. So the utility vector input for the regret minimizer of child p_i is $\boldsymbol{u}_{p_i}^{(t)}$ via the decomposition in Equation 14. □

Problem 2.3 (10 points). Apply the regret circuit for the convex hull in each decision node $p \in \mathcal{J}$. Derive the utility vector given as input to the regret minimizer $\mathfrak{R}_j$ that is responsible for mixing over $\mathcal{A}_j$, and show that it matches the utilities specified in Algorithm 1.

★ Hint: Use the decomposition given in (13).

Solution. At a decision node $j \in \mathcal{J}$, the utility of $\boldsymbol{x}_j \in \mathcal{X}_j$ is given as a convex combination of action probabilities over the columns of $\mathcal{X}_j$. Referring to Equation 13, the utility for action $a \in \mathcal{A}_j$ is given by

$$\boldsymbol{u}_j^{(t)}[a] = \langle (\boldsymbol{e}_a, \boldsymbol{0}, \cdots, \boldsymbol{x}_{\rho(j,a)}, \cdots, \boldsymbol{0}), \boldsymbol{u}_j^{(t)} \rangle.$$

Expanding the inner product, we obtain

$$\boldsymbol{u}_j^{(t)}[a] = \langle \boldsymbol{e}_a, \boldsymbol{u}_j^{(t)} \rangle + \langle \boldsymbol{x}_{\rho(j,a)}, \boldsymbol{u}_j^{(t)} \rangle.$$

Where we abuse the notation in that $\boldsymbol{u}_j^{(t)}$ is the matching block to the LHS vector. We note that the first inner product is exactly the definition of $\boldsymbol{u}^{(t)}[(j,a)]$, while the second inner product is precisely the value accrued in the child subtree $V^{(t)}[\rho(j,a)]$. Therefore, we obtain

$$\boldsymbol{u}_j^{(t)}[a] = \boldsymbol{u}^{(t)}[(j,a)] + V^{(t)}[\rho(j,a)]$$

which matches with Line 13 of Algorithm 1. The utility vector given as input for mixing over $\mathcal{A}_j$ is the

concatenation over all $a \in \mathcal{A}_j$. In other words,

$$\boldsymbol{u}_j^{(t)} = \begin{pmatrix} \boldsymbol{u}^{(t)}[(j, a_1)] + V^{(t)}[\rho(j, a_1)] \\ \boldsymbol{u}^{(t)}[(j, a_2)] + V^{(t)}[\rho(j, a_2)] \\ \vdots \\ \boldsymbol{u}^{(t)}[(j, a_\nu)] + V^{(t)}[\rho(j, a_\nu)] \end{pmatrix}$$

where $a_i \in \mathcal{A}_j$. □

Problem 2.4 (10 points). Show that the regret of `CFR` is bounded by $\sum_{j \in \mathcal{J}} \max\{0, \mathsf{Reg}_j^{(T)}\}$, where $\mathsf{Reg}_j^{(T)}$ is the regret of $\mathfrak{R}_j$. What is the regret of `CFR` when each $\mathfrak{R}_j$ is instantiated using `MWU`? What is the regret of `CFR` when each $\mathfrak{R}_j$ is instantiated using `RM`?

★ Hint: Recall that the regret bound depends on the norm of the utility; argue about how large that norm can be.

Solution. Let $\mathsf{Reg}_\Sigma^{(T)}(v)$ denote the cumulative regret at node $v \in \mathcal{J} \cup \mathcal{K}$ in the tree. Let $\mathcal{J}(v)$ denote the information sets in the sub-tree of node v. Let $*$ denote the root node of the tree. We prove

$$\mathsf{Reg}_\Sigma^{(T)}(*) \leq \sum_{j \in \mathcal{J}(*)} \max\{0, \mathsf{Reg}_j^{(T)}\} \tag{15}$$

by induction on the height of the sub-tree at v.

Base: Let the height of the sub-tree be 1. In other words, it is a terminal decision point or terminal observation node. If $v \in \mathcal{J}$ is a terminal decision point, applying the regret circuit on its convex hull obtains

$$\mathsf{Reg}_\Sigma^{(T)}(v) \leq \mathsf{Reg}_v^{(T)} + \max_{p \in C_v} \mathsf{Reg}_p^{(T)}$$

where C_v is the set of children of node v. Since v is a terminal decision point, each child p is terminal and has zero regret since no actions can be taken. Therefore, we have

$$\mathsf{Reg}_\Sigma^{(T)}(v) \leq \mathsf{Reg}_v^{(T)} + 0 \leq \max\{0, \mathsf{Reg}_v^{(T)}\} = \sum_{j \in \mathcal{J}(v)} \max\{0, \mathsf{Reg}_j^{(T)}\}$$

where we have the summation over only one element (namely v) in the information set $\mathcal{J}(v)$. Conversely, if $v \in \mathcal{K}$ is an observation node, applying the regret circuit on its Cartesian product obtains

$$\mathsf{Reg}_\Sigma^{(T)}(v) = \sum_{p \in C_v} \mathsf{Reg}_p^{(T)}$$

where C_v is the set of children of node v. Since v is a terminal observation node each child p is terminal and has zero regret since no actions can be taken. Therefore, we have

$$\mathsf{Reg}_\Sigma^{(T)}(v) = \sum_{p \in C_v} 0 = 0 \leq \sum_{j \in \mathcal{J}(v)} \max\{0, \mathsf{Reg}_j^{(T)}\}$$

where the RHS summation is a tautology since the information set $\mathcal{J}(v)$ is the null set. So for the base case, we have

$$\mathsf{Reg}_{\Sigma}^{(T)}(v) \leq \sum_{j \in \mathcal{J}(v)} \max\{0, \mathsf{Reg}_{j}^{(T)}\}.$$

Induction Hypothesis: Assume

$$\mathsf{Reg}_{\Sigma}^{(T)}(v) \leq \sum_{j \in \mathcal{J}(v)} \max\{0, \mathsf{Reg}_{j}^{(T)}\}$$

holds for all nodes v with sub-tree heights of less than the height of the entire tree.

Induction Step: For the root node $v = *$, if v is an observation node, applying the regret circuit algorithm on its Cartesian product obtains

$$\mathsf{Reg}_{\Sigma}^{(T)}(v) = \sum_{p \in C_v} \mathsf{Reg}_{p}^{(T)}.$$

Applying the inductive hypothesis on each child node p, we have

$$\mathsf{Reg}_{\Sigma}^{(T)}(v) \leq \sum_{p \in C_v} \sum_{j \in \mathcal{J}(p)} \max\{0, \mathsf{Reg}_{j}^{(T)}\} \leq \sum_{j \in \mathcal{J}(v)} \max\{0, \mathsf{Reg}_{j}^{(T)}\}$$

where the double summation collapses by observing the definition of information set $\mathcal{J}(v)$. If v is a decision node, applying the regret circuit algorithm on its convex hull obtains

$$\mathsf{Reg}_{\Sigma}^{(T)}(v) \leq \mathsf{Reg}_{v}^{(T)} + \max_{p \in C_v} \mathsf{Reg}_{p}^{(T)}.$$

Applying the inductive hypothesis on each child node p, we have

$$\mathsf{Reg}_{\Sigma}^{(T)}(v) \leq \mathsf{Reg}_{v}^{(T)} + \max_{p \in C_v} \sum_{j \in \mathcal{J}(p)} \max\{0, \mathsf{Reg}_{j}^{(T)}\}$$

Noting that $\max\{0, \mathsf{Reg}_{j}^{(T)}\}$ is non-negative, the largest sub-tree regret for a child $p \in C_v$ is less than the total sub-tree regret for all children of v. Therefore, we can further bound the above by

$$\mathsf{Reg}_{\Sigma}^{(T)}(v) \leq \mathsf{Reg}_{v}^{(T)} + \sum_{j \in \cup_{p \in C_v} \mathcal{J}(p)} \max\{0, \mathsf{Reg}_{j}^{(T)}\}. \tag{16}$$

Noting that

$$\mathsf{Reg}_{v}^{(T)} \leq \max\{0, \mathsf{Reg}_{v}^{(T)}\} = \sum_{j \in \mathcal{J}(v) \setminus \cup_{p \in C_v} \mathcal{J}(p)} \max\{0, \mathsf{Reg}_{j}^{(T)}\} \tag{17}$$

summing together Equations 16 and 17, we obtain

$$\mathsf{Reg}_{\Sigma}^{(T)}(v) \leq \sum_{j \in \mathcal{J}(v)} \max\{0, \mathsf{Reg}_{j}^{(T)}\}.$$

Since $v = *$ is the root node, we have

$$\mathsf{Reg}_{\Sigma}^{(T)}(*) \leq \sum_{j \in \mathcal{J}} \max\{0, \mathsf{Reg}_{j}^{(T)}\}$$

as $\mathcal{J} = \mathcal{J}(*)$. □

Solution. `MWU`: From Lecture 4, the regret of `MWU` is bounded as

$$\mathsf{Reg}_j^{(T)} \leq 2\sqrt{T \ln |A_j|}$$

when $\left\|\boldsymbol{u}^{(T)}\right\|_\infty \leq 1$, $\forall t$. Substituting into Equation 15, we have

$$\mathsf{Reg}_\Sigma^{(T)}(*) \leq 2 \sum_{j \in \mathcal{J}} \sqrt{T \ln |A_j|} = 2\sqrt{T} \sum_{j \in \mathcal{J}} \sqrt{\ln |A_j|}$$

`RM`: From Lecture 4, the regret of `RM` is bounded as

$$\mathsf{Reg}_j^{(T)} \leq \sqrt{T|A_j|}$$

for $(\boldsymbol{u}^{(t)})_{t=1}^T$ in $[0,1]^A$. Substituting into Equation 15, we have

$$\mathsf{Reg}_\Sigma^{(T)}(*) \leq \sum_{j \in \mathcal{J}} \sqrt{T|A_j|} = \sqrt{T} \sum_{j \in \mathcal{J}} \sqrt{|A_j|}$$

□

3 Linear programming for solving extensive-form zero-sum games, and application to low randomization in poker (50 points)

In many games, the optimal Nash equilibrium requires that all players randomize their moves. As an example, consider rock-paper-scissors: any deterministic strategy (for example, always playing rock) is heavily suboptimal. In this problem, you will quantify how much value is lost by insisting on playing deterministic strategies in three games: rock-paper-superscissors and two well-known poker variants—Kuhn poker [Kuhn, 1950] and Leduc poker [Southey et al., 2005]. A description of each game is given in the zip of this homework. The description is specified in Section 3.1. The zip of the homework also contains a stub Python file to help you set up your implementation.

3.1 Format of the game files

Each game is encoded as a json file with the following structure.

- At the root, we have a dictionary with three keys: `decision_problem_pl1`, `decision_problem_pl2`, and `utility_pl1`. The first two keys contain a description of the tree-form sequential decision problems faced by the two players, while the third is a description of the bilinear utility function for Player 1 as a function of the sequence-form strategies of each player. Since both games are zero-sum, the utility for Player 2 is the opposite of the utility of Player 1.
- The tree of decision points and observation points for each decision problem is stored as a list of nodes. Each node has the following fields:

  `id` is a string that represents the identifier of the node. The identifier is unique among the nodes for the same player.

  `type` is a string with value either `decision` (for decision points) or `observation` (for observation points).

  `actions` (only for decision points). This is a set of strings, representing the actions available at the decision node.

`signals` (only for observation points). This is a set of strings, representing the signals that can be observed at the observation node.

`parent_edge` identifies the parent edge of the node. If the node is the root of the tree, then it is `null`. Else, it is a pair `(parent_node_id, action_or_signal)`, where the first member is the `id` of the parent node, and `action_or_signal` is the action or signal that connects the node to its parent.

`parent_sequence` (only for decision points). Identifies the parent sequence p_j of the decision point, defined as the last sequence (that is, decision point-action pair) encountered on the path from the root of the decision process to j.

> **Remark 1.** The list of nodes of the tree-form sequential decision process is given in top-down traversal order. The bottom-up traversal order can be obtained by reading the list of nodes backwards.

- The bilinear utility function for Player 1 is given through the payoff matrix $\mathbf{A}$ such that the (expected) utility of Player 1 can be written as

$$u_1(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \mathbf{A}\boldsymbol{y} \rangle,$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are sequence-form strategies for Players 1 and 2 respectively. We represent $\mathbf{A}$ in the file as a list of all non-zero matrix entries, storing for each the row index, column index, and value. Specifically, each entry is an object with the fields

`sequence_pl1` is a pair `(decision_pt_id_pl1, action_pl1)` which represents the sequence of Player 1 (row of the entry in the matrix).

`sequence_pl2` is a pair `(decision_pt_id_pl2, action_pl2)` which represents the sequence of Player 2 (column of the entry in the matrix).

`value` is the non-zero float value of the matrix entry.

Example: Rock-paper-superscissors In the case of rock-paper-superscissors the decision problem faced by each of the players has only one decision points with three actions: playing rock, paper, or superscissors. So, each tree-form sequential decision process only has a single node, which is a decision node. The payoff matrix of the game is

$$\begin{array}{c} \\ \mathrm{r} \\ \mathrm{p} \\ \mathrm{s} \end{array} \begin{array}{c} \begin{array}{ccc} \mathrm{r} & \mathrm{p} & \mathrm{s} \end{array} \\ \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -2 \\ -1 & 2 & 0 \end{pmatrix} \end{array}.$$

So, the game file in this case has content:

```
{
  "decision_problem_pl1": [
    {"id": "d1_pl1", "type": "decision", "actions": ["r", "p", "s"],
     "parent_edge": null, "parent_sequence": null}
  ],
  "decision_problem_pl2": [
    {"id": "d1_pl2", "type": "decision", "actions": ["r", "p", "s"],
     "parent_edge": null, "parent_sequence": null}
  ],
  "utility_pl1": [
    {"sequence_pl1": ["d1_pl1", "r"], "sequence_pl2": ["d1_pl2", "p"], "value": -1},
```

```
    {"sequence_pl1": ["d1_pl1", "r"], "sequence_pl2": ["d1_pl2", "s"], "value": 1},
    {"sequence_pl1": ["d1_pl1", "p"], "sequence_pl2": ["d1_pl2", "r"], "value": 1},
    {"sequence_pl1": ["d1_pl1", "p"], "sequence_pl2": ["d1_pl2", "s"], "value": -2},
    {"sequence_pl1": ["d1_pl1", "s"], "sequence_pl2": ["d1_pl2", "r"], "value": -1},
    {"sequence_pl1": ["d1_pl1", "s"], "sequence_pl2": ["d1_pl2", "p"], "value": 2}
  ]
}
```


3.2 Computing the value of the game (15 points)

As a warm up, you will implement the linear program formulation of Nash equilibrium strategies seen in Lecture 5 using the commercial solver Gurobi (https://www.gurobi.com/). Gurobi is a powerful commercial solver for linear and non-linear optimization problems. You can download the solver and request a free license for academic use from their website.

Installing `gurobipy` Installation instructions for Gurobi's python bindings are available on the Gurobi website, here.[1]

Linear programming formulation of Nash equilibrium strategies For your convenience, here are again the linear programs—for Player 1 and Player 2, respectively—that you need to implement:

$$\mathcal{P}_1 : \begin{cases} \max \ \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{s.t.} \ \ ① \ \mathbf{A}^\top \boldsymbol{x} - \mathbf{F}_2^\top \boldsymbol{v} \geq \mathbf{0} \\ \quad\ \ ② \ \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ \quad\ \ ③ \ \boldsymbol{x} \geq \mathbf{0}, \ \boldsymbol{v} \text{ free}, \end{cases} \qquad \mathcal{P}_2 : \begin{cases} \max \ \boldsymbol{f}_1^\top \boldsymbol{v} \\ \text{s.t.} \ \ ① \ -\mathbf{A}\boldsymbol{y} - \mathbf{F}_1^\top \boldsymbol{v} \geq \mathbf{0} \\ \quad\ \ ② \ \mathbf{F}_2 \boldsymbol{y} = \boldsymbol{f}_2 \\ \quad\ \ ③ \ \boldsymbol{y} \geq \mathbf{0}, \ \boldsymbol{v} \text{ free}, \end{cases} \tag{18}$$

where $\{\boldsymbol{x} \in \mathbb{R}^{|\Sigma_1|} : \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1, \boldsymbol{x} \geq \mathbf{0}\}$ and $\{\boldsymbol{y} \in \mathbb{R}^{|\Sigma_2|} : \mathbf{F}_2 \boldsymbol{y} = \boldsymbol{f}_2, \boldsymbol{y} \geq \mathbf{0}\}$ are the sequence-form polytopes of the two players, and $\mathbf{A}$ is the payoff matrix for Player 1. Conveniently, the objective values of $\mathcal{P}_1$ and $\mathcal{P}_2$ will be the exact expected utility that each player can secure by playing against a perfectly rational opponent. Since all games are zero sum, the objective values of $\mathcal{P}_1$ and $\mathcal{P}_2$ will sum to 0 (if they don't, you must have a bug somewhere).

Problem 3.1 (15 points). Implement the linear program for computing Nash equilibrium strategies for both Player 1 and Player 2.

For each of the three games (rock-paper-superscissors, Kuhn poker, and Leduc poker), and for each of the two player, report Gurobi's output.

★ Hint: make sure to take a look at the "Gurobi tips and tricks" at the end of this document. It includes some tips as to how to debug common issues.
★ Hint: start from rock-paper-superscissors, and only then move to the more complex games.
★ Hint: since all games are zero-sum, the objective values of $\mathcal{P}_1$ and $\mathcal{P}_2$ must sum to 0.
★ Hint: the objective value for $\mathcal{P}_1$ should be 0 for rock-paper-superscissors, -0.0555 for Kuhn poker, and -0.0856 for Leduc poker.

Solution. Gurobi output for rock-paper-superscissors (player 1 solution first, followed by player 2):

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads
```

[1] https://www.gurobi.com/documentation/9.1/quickstart_linux/cs_python.html#section:Python

```
Optimize a model with 4 rows, 4 columns and 12 nonzeros
Model fingerprint: 0x8189a642
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 1 rows and 0 columns
Presolve time: 0.02s
Presolved: 3 rows, 4 columns, 11 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0000000e+00   1.000000e+00   0.000000e+00      0s
       2   -0.0000000e+00   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.04 seconds (0.00 work units)
Optimal objective -0.000000000e+00
Player 1 Objective: -0.0
---------------------------------------------------------------------
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 4 rows, 4 columns and 12 nonzeros
Model fingerprint: 0x8189a642
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 1 rows and 0 columns
Presolve time: 0.00s
Presolved: 3 rows, 4 columns, 11 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0000000e+00   1.000000e+00   0.000000e+00      0s
       2   -0.0000000e+00   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.00 seconds (0.00 work units)
Optimal objective -0.000000000e+00
Player 2 Objective: -0.0
```

Gurobi output for Kuhn poker (player 1 solution first, followed by player 2):

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads
```

```
Optimize a model with 18 rows, 18 columns and 57 nonzeros
Model fingerprint: 0x183a00fa
Coefficient statistics:
  Matrix range     [2e-01, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 11 rows and 5 columns
Presolve time: 0.02s
Presolved: 7 rows, 13 columns, 33 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.6666667e-01   6.666667e-01   0.000000e+00      0s
       5   -5.5555556e-02   0.000000e+00   0.000000e+00      0s

Solved in 5 iterations and 0.03 seconds (0.00 work units)
Optimal objective -5.555555556e-02
Player 1 Objective: -0.055555555555555525
---------------------------------------------------------------------
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 18 rows, 18 columns and 57 nonzeros
Model fingerprint: 0x4696a259
Coefficient statistics:
  Matrix range     [2e-01, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 15 rows and 13 columns
Presolve time: 0.00s
Presolved: 3 rows, 5 columns, 7 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.6666667e-01   3.333333e-01   0.000000e+00      0s
       2    5.5555556e-02   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.00 seconds (0.00 work units)
Optimal objective  5.555555556e-02
Player 2 Objective: 0.055555555555555525
```

Gurobi output for Leduc poker (player 1 solution first, followed by player 2):

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads
```

```
Optimize a model with 480 rows, 480 columns and 1917 nonzeros
Model fingerprint: 0xd72e9747
Coefficient statistics:
  Matrix range     [3e-02, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 255 rows and 156 columns
Presolve time: 0.02s
Presolved: 225 rows, 324 columns, 2283 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0      handle free variables                         0s
     357   -8.5606424e-02   0.000000e+00   0.000000e+00      0s

Solved in 357 iterations and 0.05 seconds (0.01 work units)
Optimal objective -8.560642408e-02
Player 1 Objective: -0.08560642407800051
---------------------------------------------------------------------
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 480 rows, 480 columns and 1917 nonzeros
Model fingerprint: 0x655f2f05
Coefficient statistics:
  Matrix range     [3e-02, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 151 rows and 144 columns
Presolve time: 0.00s
Presolved: 329 rows, 336 columns, 2314 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0      handle free variables                         0s
     295    8.5606424e-02   0.000000e+00   0.000000e+00      0s

Solved in 295 iterations and 0.01 seconds (0.01 work units)
Optimal objective  8.560642408e-02
Player 2 Objective: 0.08560642407800045
```

□

3.3 Computing optimal deterministic strategies (15 points)

In this subsection we study how much worse each player is if they (but not the opponent) are restricted to playing deterministic strategies only. To model this, we will add a constraint saying that all entries of the sequence-form strategy vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in (18) can only take values in $\{0, 1\}$. The resulting *integer*

programs—which we call $\tilde{\mathcal{P}}_1$ and $\tilde{\mathcal{P}}_2$—are given next.

$$\tilde{\mathcal{P}}_1 : \begin{cases} \max \ \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{s.t.} \ \ ① \ \mathbf{A}^\top \boldsymbol{x} - \mathbf{F}_2^\top \boldsymbol{v} \geq \mathbf{0} \\ \qquad ② \ \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ \qquad ③ \ \boldsymbol{x} \in \{0,1\}^{|\Sigma_1|}, \ \boldsymbol{v} \text{ free}, \end{cases} \qquad \tilde{\mathcal{P}}_2 : \begin{cases} \max \ \boldsymbol{f}_1^\top \boldsymbol{v} \\ \text{s.t.} \ \ ① \ -\mathbf{A}\boldsymbol{y} - \mathbf{F}_1^\top \boldsymbol{v} \geq \mathbf{0} \\ \qquad ② \ \mathbf{F}_2 \boldsymbol{y} = \boldsymbol{f}_2 \\ \qquad ③ \ \boldsymbol{y} \in \{0,1\}^{|\Sigma_2|}, \ \boldsymbol{v} \text{ free}. \end{cases} \tag{19}$$

Problem 3.2 (15 points). Implement the integer programs given in (19) for computing optimal deterministic strategies for both Player 1 and Player 2.

For each of the three games (rock-paper-superscissors, Kuhn poker, and Leduc poker), and for each of the two player, report Gurobi's output.

★ Hint: make sure to take a look at the "Gurobi tips and tricks" at the end of this document. It includes some tips as to how to debug common issues.
★ Hint: start from rock-paper-superscissors, and only then move to the more complex games.
★ Hint: here there are *no guarantees* that the value of $\tilde{\mathcal{P}}_1$ and the value of $\tilde{\mathcal{P}}_2$ sum to 0 anymore! In fact, that will be false in all games.

Solution. Gurobi output for rock-paper-superscissors (player 1 solution first, followed by player 2):

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 4 rows, 4 columns and 12 nonzeros
Model fingerprint: 0x482b06ee
Variable types: 1 continuous, 3 integer (3 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 4 rows and 4 columns
Presolve time: 0.03s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.04 seconds (0.00 work units)
Thread count was 1 (of 10 available processors)

Solution count 1: -1
No other solutions better than -1

Optimal solution found (tolerance 1.00e-04)
Best objective -1.000000000000e+00, best bound -1.000000000000e+00, gap 0.0000%
Player 1 Objective: -1.0
------------------------------------------------------------------------
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
```

```
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 4 rows, 4 columns and 12 nonzeros
Model fingerprint: 0x482b06ee
Variable types: 1 continuous, 3 integer (3 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 4 rows and 4 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 10 available processors)

Solution count 1: -1
No other solutions better than -1

Optimal solution found (tolerance 1.00e-04)
Best objective -1.000000000000e+00, best bound -1.000000000000e+00, gap 0.0000%
Player 2 Objective: -1.0
```

Gurobi output for Kuhn poker (player 1 solution first, followed by player 2):

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 18 rows, 18 columns and 57 nonzeros
Model fingerprint: 0xfc1ab629
Variable types: 6 continuous, 12 integer (12 binary)
Coefficient statistics:
  Matrix range     [2e-01, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.3333333
Presolve removed 18 rows and 18 columns
Presolve time: 0.03s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.04 seconds (0.00 work units)
Thread count was 1 (of 10 available processors)

Solution count 2: -0.166667 -0.333333
No other solutions better than -0.166667
```

```
Optimal solution found (tolerance 1.00e-04)
Best objective -1.666666666667e-01, best bound -1.666666666667e-01, gap 0.0000%
Player 1 Objective: -0.16666666666666666
-----------------------------------------------------------------
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 18 rows, 18 columns and 57 nonzeros
Model fingerprint: 0x3caf3179
Variable types: 6 continuous, 12 integer (12 binary)
Coefficient statistics:
  Matrix range     [2e-01, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -0.3333333
Presolve removed 18 rows and 18 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 10 available processors)

Solution count 2: -0.166667 -0.333333
No other solutions better than -0.166667

Optimal solution found (tolerance 1.00e-04)
Best objective -1.666666666667e-01, best bound -1.666666666667e-01, gap 0.0000%
Player 2 Objective: -0.16666666666666669
```

Gurobi output for Leduc poker (player 1 solution first, followed by player 2):

```
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads

Optimize a model with 480 rows, 480 columns and 1917 nonzeros
Model fingerprint: 0x448ac102
Variable types: 144 continuous, 336 integer (336 binary)
Coefficient statistics:
  Matrix range     [3e-02, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 102 rows and 102 columns
Presolve time: 0.05s
Presolved: 378 rows, 378 columns, 1742 nonzeros
```

```
Variable types: 126 continuous, 252 integer (252 binary)
Found heuristic solution: objective -2.0000000
Found heuristic solution: objective -1.5000000

Root relaxation: objective -1.521596e-01, 512 iterations, 0.01 seconds (0.01 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   -0.15216    0   64   -1.50000   -0.15216  89.9%     -    0s
H    0     0                       -1.1000000   -0.15216  86.2%     -    0s
H    0     0                       -0.7666667   -0.15216  80.2%     -    0s
H    0     0                       -0.7000000   -0.19032  72.8%     -    0s
     0     0   -0.20278    0   69   -0.70000   -0.20278  71.0%     -    0s
H    0     0                       -0.6000000   -0.20278  66.2%     -    0s
H    0     0                       -0.4333333   -0.20278  53.2%     -    0s
H    0     0                       -0.4000000   -0.21300  46.8%     -    0s
     0     0   -0.21300    0   69   -0.40000   -0.21300  46.8%     -    0s
     0     0   -0.24335    0   71   -0.40000   -0.24335  39.2%     -    0s
H    0     0                       -0.3000000   -0.24335  18.9%     -    0s
     0     0   -0.24635    0   57   -0.30000   -0.24635  17.9%     -    0s
     0     0   -0.25545    0   74   -0.30000   -0.25545  14.8%     -    0s
     0     0   -0.26667    0   72   -0.30000   -0.26667  11.1%     -    0s
     0     0   -0.26667    0   79   -0.30000   -0.26667  11.1%     -    0s
     0     0   -0.27023    0   41   -0.30000   -0.27023  9.92%     -    0s
     0     0   -0.27023    0   61   -0.30000   -0.27023  9.92%     -    0s
     0     0   -0.27023    0   61   -0.30000   -0.27023  9.92%     -    0s

Cutting planes:
  Gomory: 4
  Implied bound: 1
  MIR: 24
  RLT: 1
  Relax-and-lift: 3

Explored 1 nodes (1024 simplex iterations) in 0.29 seconds (0.08 work units)
Thread count was 10 (of 10 available processors)

Solution count 10: -0.3 -0.3 -0.4 ... -1.1
No other solutions better than -0.3

Optimal solution found (tolerance 1.00e-04)
Best objective -3.000000000000e-01, best bound -3.000000000000e-01, gap 0.0000%
Player 1 Objective: -0.3
------------------------------------------------------------------
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[rosetta2] - Darwin 23.5.0 23F79)

CPU model: Apple M1 Pro
Thread count: 10 physical cores, 10 logical processors, using up to 10 threads
```

```
Optimize a model with 480 rows, 480 columns and 1917 nonzeros
Model fingerprint: 0x1fa9e708
Variable types: 144 continuous, 336 integer (336 binary)
Coefficient statistics:
  Matrix range     [3e-02, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective -2.3666667
Presolve removed 104 rows and 104 columns
Presolve time: 0.01s
Presolved: 376 rows, 376 columns, 1705 nonzeros
Found heuristic solution: objective -1.3333333
Variable types: 88 continuous, 288 integer (288 binary)

Root relaxation: objective -1.478962e-02, 436 iterations, 0.01 seconds (0.01 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0   -0.01479    0   90   -1.33333   -0.01479  98.9%     -    0s
H    0     0                      -0.8000000   -0.01479  98.2%     -    0s
H    0     0                      -0.5333333   -0.07893  85.2%     -    0s
     0     0   -0.08327    0  134   -0.53333   -0.08327  84.4%     -    0s
H    0     0                      -0.3666667   -0.08327  77.3%     -    0s
     0     0   -0.11382    0  132   -0.36667   -0.11382  69.0%     -    0s
     0     0   -0.18127    0  144   -0.36667   -0.18127  50.6%     -    0s
     0     0   -0.18371    0  138   -0.36667   -0.18371  49.9%     -    0s
     0     2   -0.18371    0  138   -0.36667   -0.18371  49.9%     -    0s
H   42    46                      -0.2666667   -0.18371  31.1%  19.9    0s

Cutting planes:
  Gomory: 10
  Implied bound: 1
  MIR: 31

Explored 148 nodes (2539 simplex iterations) in 0.15 seconds (0.12 work units)
Thread count was 10 (of 10 available processors)

Solution count 6: -0.266667 -0.366667 -0.533333 ... -2.36667
No other solutions better than -0.266667

Optimal solution found (tolerance 1.00e-04)
Best objective -2.666666666667e-01, best bound -2.666666666666e-01, gap 0.0000%
Player 2 Objective: -0.26666666666666666
```

□

3.4 Controlling the amount of determinism (20 points)

In Problem 3.1 no determinism constraint was present. At the other extreme, in Problem 3.2 we insisted that at all decision points a deterministic strategy be followed. In this last subsection we will explore intermediate cases: for each value of k, we will study how much value each player can secure if they are constrained to play deterministically in at least k decision points. When $k = 0$, we will recover the objective values seen in Problem 3.1. When k is equal to the number of decision points of the player in the game, we will recover the objective values seen in Problem 3.2.

Integer programming model An optimal strategy for Player 1 subject to the constraint that at least k decision points must prescribe a deterministic strategy can be obtained as the solution to the integer linear program $\mathcal{P}_1(k)$ given in (20). Understanding the details is not important for this problem, though we included a description of the meaning of each constraint just in case you are curious.

$$\mathcal{P}_1(k): \begin{cases} \max & \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{s.t.} & \textcircled{1}\ \mathbf{A}^\top \boldsymbol{x} - \mathbf{F}_2^\top \boldsymbol{v} \geq \mathbf{0} \\ & \textcircled{2}\ \mathbf{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ & \textcircled{3}\ \boldsymbol{x}[(j,a)] \geq \boldsymbol{z}[(j,a)] & \forall j \in \mathcal{J}_1 : p_j = \varnothing, \quad a \in \mathcal{A}_j \\ & \textcircled{4}\ \boldsymbol{x}[(j,a)] \geq \boldsymbol{x}[p_j] + \boldsymbol{z}[(j,a)] - 1 & \forall j \in \mathcal{J}_1 : p_j \neq \varnothing, \quad a \in \mathcal{A}_j \\ & \textcircled{5}\ \sum_{a \in \mathcal{A}_j} \boldsymbol{z}[(j,a)] \leq 1 & \forall\, j \in \mathcal{J}_1 \\ & \textcircled{6}\ \sum_{j \in \mathcal{J}_1} \sum_{a \in \mathcal{A}_j} \boldsymbol{z}[(j,a)] \geq k \\ & \textcircled{7}\ \boldsymbol{x} \geq \mathbf{0},\ \boldsymbol{z} \in \{0,1\}^{|\Sigma_1|},\ \boldsymbol{v} \text{ free}, \end{cases} \tag{20}$$

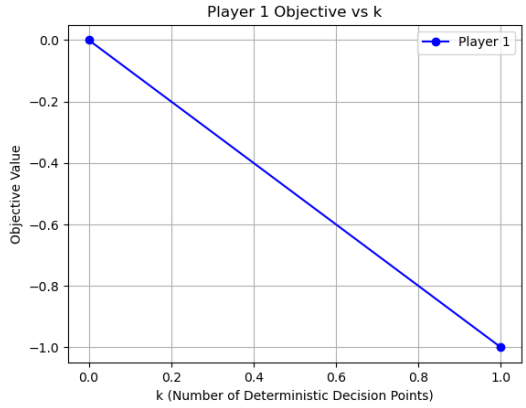
- $\boldsymbol{z} \in \{0,1\}^{|\Sigma_1|}$ is a binary vector that decides, for each strategy $(j,a) \in \Sigma_1$ of Player 1, whether to pick action a at j with probability 1. Since the strategy vector $\boldsymbol{x}$ is expressed in sequence-form, picking action a with probability 1 at j is expressed through constraints ③ and ④.
- Constraint ⑤ asserts that no more than one action at each decision point can be forced to be played with probability 1.
- Constraint ⑥ asserts that in at least k decision point, exactly one of the actions will be forced to be played with probability 1.

The integer program $\mathcal{P}_2(k)$ for Player 2 is analogous.

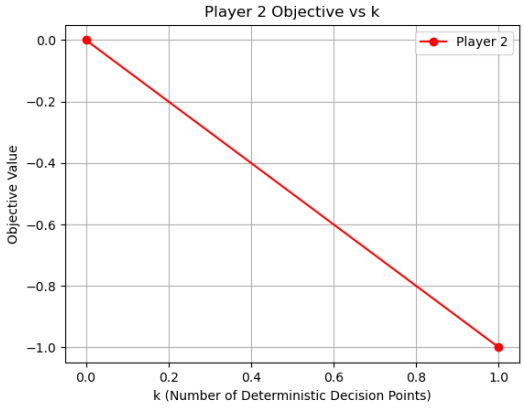
Problem 3.3 (15 points). Implement the integer linear programs $\mathcal{P}_1(k)$ and $\mathcal{P}_2(k)$, described above, for computing optimal strategies with a given lower bound on the amount of determinism.

For each of the three games (rock-paper-superscissors, Kuhn poker, and Leduc poker), and for each of the two player i, plot the objective value of $\mathcal{P}_i(k)$ as a function of $k \in \{0, \ldots, |\mathcal{J}_i|\}$ (number of decision points of Player i).

★ Hint: make sure to take a look at the "Gurobi tips and tricks" at the end of this document. It includes some tips as to how to debug common issues.
★ Hint: Gurobi can be pretty verbose by default. For this problem, if you would like to silence Gurobi you can use `m.setParam("OutputFlag", 0)`
★ Hint: For Leduc poker, if Gurobi is taking too long to optimize when k is large, you can lower the solution precision by calling `m.setParam("MIPGap", 0.01)` before `m.optimize()`. Expect a runtime of up to one-five hours for Leduc poker, depending on how powerful the machine you are using is.

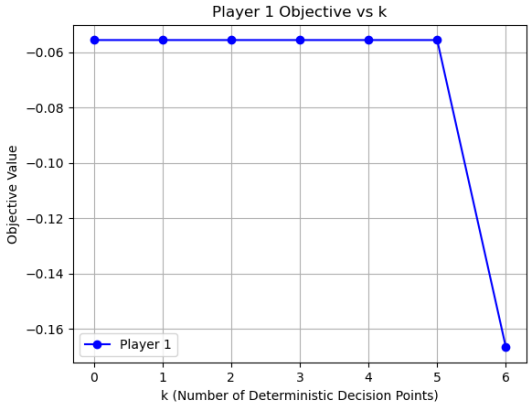


(a) Player 1

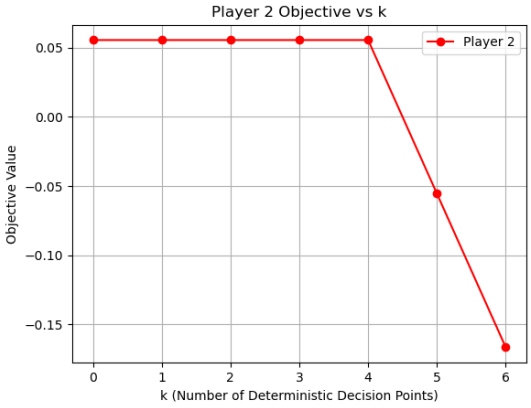


(b) Player 2

Figure 1: Plots for both players of rock-paper-superscissors

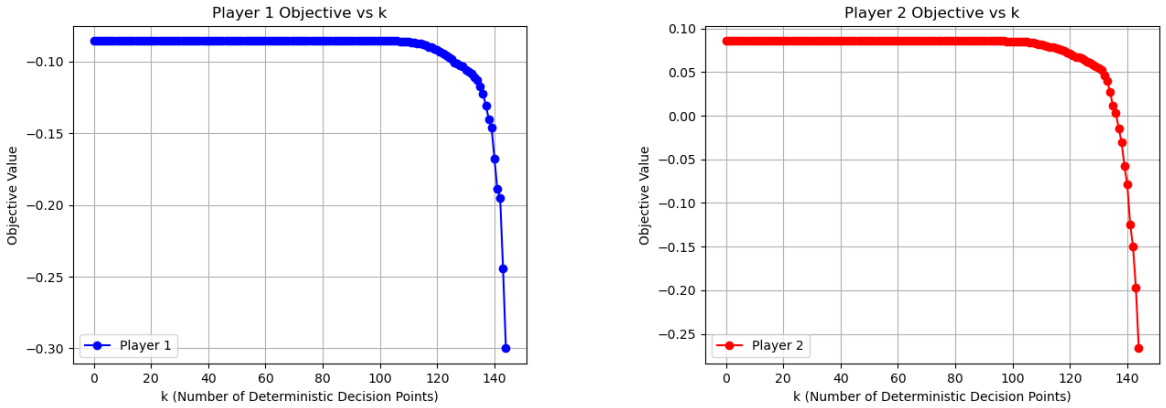


(a) Player 1



(b) Player 2

Figure 2: Plots for both players of Kuhn poker



(a) Player 1 (b) Player 2

Figure 3: Plots for both players of Leduc poker

Solution. □

Problem 3.4 (5 points). Comment on the results you obtained in this problem: do highly-deterministic strategies for the three small games exist? Are the results what you expected? If yes, what did he results confirm? If not, how do you think you can reconcile your previous intuition with the experimental findings?

Solution. With the exception of rock-paper-superscissors, highly-deterministic strategies that obtains an objective value similar to the ones obtained for Problem 3.1 exists. In the Kuhn poker example, both players can fix 4 decision points (player 1 can fix 5) to be deterministic without any effect on their objective values. In the Leduc poker example, both players can have about 100 deterministic decision points before their objective values materially degrade. For the rock-paper-superscissors example, there is only one decision point, so a highly-deterministic strategy does not exist (unless the player wants to lose every round).

The results for the poker examples is not what I expected. My previous intuition was that if you play 70% or 80% of your decision points deterministically, your opponent could surely exploit this and decrease your payoff/objective value. However from this exercise, we see that this not the case. You can still obtain your Nash equilibrium objective values even if you fix 70% or 80% of your decision points. To reconcile this, I think that this occurs is perhaps due to which decision points you decide to fix deterministically. If the fixed decision points are all off-equilibrium paths, then perhaps it is okay to fix the majority of them to be deterministic. The stochasticity in the 20%-ish on-equilibrium paths that gets played often would be the ones that actually mattered in optimizing the objective value.

□

A Appendix: Gurobi tips and tricks

Basic notation Let `m` denote the Gurobi model object. Then, here is a quick cookbook.

- Add a continuous free variable:
  `m.addVar(-GRB.INFINITY, GRB.INFINITY, vtype=GRB.CONTINUOUS, name="human_var_name_here")`

- Add a continuous nonnegative variable:
  `m.addVar(0.0, GRB.INFINITY, vtype=GRB.CONTINUOUS, name="human_var_name_here")`
- Add a binary variable:
  `m.addVar(0.0, 1.0, vtype=GRB.BINARY, name="human_var_name_here")`
- Add an equality constraint:
  `m.addConstr(lhs == rhs)`
- Add an inequality ($\geq$) constraint:
  `m.addConstr(lhs >= rhs)`
- Set a maximization objective:
  `m.setObjective(obj, sense=GRB.MAXIMIZE)`

Accessing the solution After calling `m.optimize()`, you can obtain the objective value by calling

```
m.getAttr(GRB.Attr.ObjVal)
```

If you want to inspect the solution, given a variable object `v` (the object returned by `m.addVar`), you can access the value of `v` in the current solution by calling

```
v.getAttr(GRB.Attr.X)
```

Troubleshooting First of all, if you are having a problem with Gurobi, the first thing you should try to do is to ask Gurobi to dump the model that it thinks you are asking to solve to a file in a human readable format. *Reading the model file will be so much easier if you gave names to the variables in your model, using the '`name`' optional argument of `addVar`.*

To have Gurobi dump the model, you can use something like this:

```
m.write("/tmp/model.lp")
```

Of course, you can specify a different path. However, it is important that you keep the '`.lp`' extension: there are multiple format that Gurobi can output, and it uses the file extension to guess which format you want.

Beyond the general rule of thumb above, make sure of the following:

- Start from rock-paper-superscissors. There, the `/tmp/model.lp` file for Player 1 for Problem 3.1 should look something like this (probably with different variable names):

```
\ Model game_value_pl1
\ LP format - for model browsing. Use MPS format to capture full model detail.
Maximize
    v[d1_pl2]
Subject To
    R0: x[('d1_pl1','p')] - x[('d1_pl1','s')] - v[d1_pl2] >= 0
    R1: - x[('d1_pl1','r')] + 2 x[('d1_pl1','s')] - v[d1_pl2] >= 0
    R2: x[('d1_pl1','r')] - 2 x[('d1_pl1','p')] - v[d1_pl2] >= 0
    R3: x[('d1_pl1','r')] + x[('d1_pl1','p')] + x[('d1_pl1','s')] = 1
Bounds
    v[d1_pl2] free
End
```

  Note: Gurobi omits listing nonnegative variables in the `Bounds` section.

- Did you remember to specify that you want a *maximization* problem? (Gurobi's default is minimization) If Gurobi says that the model is unbounded, chances are you forgot.
- Check that the number of variables and constraints is what you expect. Are the sense of the constraints (equality, $\leq$, $\geq$) what you wanted?

B Code

```
#!/usr/bin/env python3

import os
import argparse
import json
from collections import defaultdict
import matplotlib.pyplot as plt
import gurobipy as gurobi
from gurobipy import GRB


def build_strategy_polytope(tfsdp):

    # Build decision points
    J = [] # Decision nodes
    A_j = {} # Action set at node j
    p_j = {} # Parent sequence at node j

    for node in tfsdp:
        if node["type"] == "decision":
            j = node["id"]
            J.append(j)
            A_j[j] = list(node["actions"])
            p_j[j] = node.get("parent_sequence", None)

    # Build all sequences of (j, a) avaliable to player
    sequences = []
    for j in J:
        for a in A_j[j]:
            sequences.append((j, a))
    sequence_index = {s: idx for idx, s in enumerate(sequences)}

    # Build matrix representation of strategy polytope
    F = [] # Matrix of shape |J| x |sequences| (each row is represented as a dict)
    f = [] # Vector of shape |J|

    for j in J:
        row = defaultdict(float)
        for a in A_j[j]:
            row[sequence_index[(j, a)]] += 1.0

        parent_sequence = p_j[j]
        if parent_sequence is None:
            f.append(1.0)
        else:
            row[sequence_index[parent_sequence]] -= 1.0
            f.append(0.0)

        F.append(row)
    return sequences, F, f, J, A_j, p_j


def build_payoff_matrix(game, S1, S2):

    # Build payoff matrix of shape |S1| x |S2| in terms of player 1's utility (only store non-zero entres using a dict)
    A = defaultdict(float)
    r_idx = {s: i for i, s in enumerate(S1)}
    c_idx = {s: j for j, s in enumerate(S2)}

    for leaf_node in game["utility_pl1"]:
        s1 = leaf_node["sequence_pl1"]
        s2 = leaf_node["sequence_pl2"]
```

```
        value = float(leaf_node["value"])

        if s1 in r_idx and s2 in c_idx:
            A[(r_idx[s1], c_idx[s2])] += value
    return A


def build_all_matrices(game):

    S1, F1, f1, J1, A1_j, p1_j = build_strategy_polytope(game["decision_problem_pl1"])
    S2, F2, f2, J2, A2_j, p2_j = build_strategy_polytope(game["decision_problem_pl2"])
    A = build_payoff_matrix(game, S1, S2)
    return S1, F1, f1, S2, F2, f2, A, J1, A1_j, p1_j, J2, A2_j, p2_j


def solve_problem_2_3_with_k(player, k, S1, F1, f1, S2, F2, f2, A, J1, A1_j, p1_j, J2, A2_j, p2_j):

    n1, n2 = len(S1), len(S2)
    m1, m2 = len(F1), len(F2)

    m = gurobi.Model("game_value_pl" + str(player) + "_" + str(k))

    if player == 1:

        # Player 1 free variables
        x = m.addVars(n1, lb=0.0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="x")
        v = m.addVars(m2, lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="v")
        z = m.addVars(n1, vtype=GRB.BINARY, name="z")

        # Constraint F1.x = f1
        for r_idx, row in enumerate(F1):
            expr = gurobi.LinExpr()
            for c_idx, coeff in row.items():
                expr.addTerms(coeff, x[c_idx])
            m.addConstr(expr == f1[r_idx], name=f"F1.x[{r_idx}]")

        # Constraint A^T.x - F2^T.v >= 0
        for j in range(n2):
            lhs = gurobi.LinExpr()

            for (r_idx, c_idx), value in A.items():
                if c_idx == j:
                    lhs.addTerms(value, x[r_idx])

            for r_idx, row in enumerate(F2):
                coeff = row.get(j, 0.0)
                if coeff:
                    lhs.addTerms(-coeff, v[r_idx])

            rhs = 0.0
            m.addConstr(lhs >= rhs, name=f"A^T.x-F2^T.v[{j}]")

        # Constraint x[(j,a)] >= z[(j,a)] for p1_j[j] == None
        for j in J1:
            if p1_j[j] is None:
                for a in A1_j[j]:
                    seq_idx = S1.index((j, a))
                    m.addConstr(x[seq_idx] >= z[seq_idx], name=f"x[(j,a)]>=z[(j,a)][{(j,a)}]")

        # Constraint x[(j,a)] >= x[p_j] + z[(j,a)] - 1 for p1_j[j] != None
        for j in J1:
            if p1_j[j] is not None:
                parent_idx = S1.index(p1_j[j])
                for a in A1_j[j]:
                    seq_idx = S1.index((j, a))
```

```
                m.addConstr(x[seq_idx] - x[parent_idx] - z[seq_idx] >= -1.0, name=f"x[(j,a)]>=x[p_j]+z[(j,a)]-1[{(j,a)}]")

    # Constraint sum_a z[(j,a)] <= 1
    for j in J1:
        seqs = [S1.index((j, a)) for a in A1_j[j]]
        m.addConstr(gurobi.quicksum(z[i] for i in seqs) <= 1)

    # Constraint sum_j sum_a z[(j,a)] >= k
    m.addConstr(gurobi.quicksum(z[i] for i in range(n1)) >= k)

    # Objective max f2^T.v
    obj = gurobi.LinExpr()
    for r_idx in range(m2):
        obj.addTerms(f2[r_idx], v[r_idx])

    m.setObjective(obj, sense=GRB.MAXIMIZE)

else:

    # Player 2 free variables
    y = m.addVars(n2, lb=0.0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="y")
    v = m.addVars(m1, lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="v")
    z = m.addVars(n2, vtype=GRB.BINARY, name="z")

    # Constraint F2.y = f2
    for r_idx, row in enumerate(F2):
        expr = gurobi.LinExpr()
        for c_idx, coeff in row.items():
            expr.addTerms(coeff, y[c_idx])
        m.addConstr(expr == f2[r_idx], name=f"F2.y[{r_idx}]")

    # Constraint -A.y - F1^T.v >= 0
    for i in range(n1):
        lhs = gurobi.LinExpr()

        for (r_idx, c_idx), value in A.items():
            if r_idx == i:
                lhs.addTerms(-value, y[c_idx])

        for r_idx, row in enumerate(F1):
            coeff = row.get(i, 0.0)
            if coeff:
                lhs.addTerms(-coeff, v[r_idx])

        rhs = 0.0
        m.addConstr(lhs >= rhs, name=f"-A.y-F1^T.v[{i}]")

    # Constraint y[(j,a)] >= z[(j,a)] for p2_j[j] == None
    for j in J2:
        if p2_j[j] is None:
            for a in A2_j[j]:
                seq_idx = S2.index((j, a))
                m.addConstr(y[seq_idx] >= z[seq_idx], name=f"y[(j,a)]>=z[(j,a)][{(j,a)}]")

    # Constraint y[(j,a)] >= y[p_j] + z[(j,a)] - 1 for p2_j[j] != None
    for j in J2:
        if p2_j[j] is not None:
            parent_idx = S2.index(p2_j[j])
            for a in A2_j[j]:
                seq_idx = S2.index((j, a))
                m.addConstr(y[seq_idx] - y[parent_idx] - z[seq_idx] >= -1.0, name=f"y[(j,a)]>=y[p_j]+z[(j,a)]-1[{(j,a)}]")

    # Constraint sum_a z[(j,a)] <= 1
    for j in J2:
        seqs = [S2.index((j, a)) for a in A2_j[j]]
```

```
            m.addConstr(gurobi.quicksum(z[i] for i in seqs) <= 1)

        # Constraint sum_j sum_a z[(j,a)] >= k
        m.addConstr(gurobi.quicksum(z[i] for i in range(n2)) >= k)

        # Objective max f1^T.v
        obj = gurobi.LinExpr()
        for r_idx in range(m1):
            obj.addTerms(f1[r_idx], v[r_idx])

        m.setObjective(obj, sense=GRB.MAXIMIZE)

    m.optimize()
    return m


def plot_objectives(player, k_values, obj_values):

    color = 'blue' if player == 1 else 'red'
    plt.figure()
    plt.plot(k_values, obj_values, marker='o', color=color, label=f"Player {player}")
    plt.title(f"Player {player} Objective vs k")
    plt.xlabel("k (Number of Deterministic Decision Points)")
    plt.ylabel("Objective Value")
    plt.grid(True)
    plt.legend()
    plt.show()


def solve_problem_2_1(game):

    S1, F1, f1, S2, F2, f2, A, _, _, _, _, _, _ = build_all_matrices(game)
    n1, n2 = len(S1), len(S2)
    m1, m2 = len(F1), len(F2)

    for player in [1, 2]:

        m = gurobi.Model("game_value_pl" + str(player))

        if player == 1:

            # Player 1 free variables
            # Reference: https://docs.gurobi.com/projects/optimizer/en/current/reference/python/model.html
            x = m.addVars(n1, lb=0.0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="x")
            v = m.addVars(m2, lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="v")

            # Constraint F1.x = f1
            # Reference: https://docs.gurobi.com/projects/optimizer/en/current/reference/python/linexpr.html
            for r_idx, row in enumerate(F1):
                expr = gurobi.LinExpr()
                for c_idx, coeff in row.items():
                    expr.addTerms(coeff, x[c_idx])
                m.addConstr(expr == f1[r_idx], name=f"F1.x[{r_idx}]")

            # Constraint A^T.x - F2^T.v >= 0
            for j in range(n2):
                lhs = gurobi.LinExpr()

                for (r_idx, c_idx), value in A.items():
                    if c_idx == j:
                        lhs.addTerms(value, x[r_idx])

                for r_idx, row in enumerate(F2):
                    coeff = row.get(j, 0.0)
                    if coeff:
```

```
                    lhs.addTerms(-coeff, v[r_idx])

                rhs = 0.0
                m.addConstr(lhs >= rhs, name=f"A^T.x-F2^T.v[{j}]")

            # Objective max f2^T.v
            obj = gurobi.LinExpr()
            for r_idx in range(m2):
                obj.addTerms(f2[r_idx], v[r_idx])

            m.setObjective(obj, sense=GRB.MAXIMIZE)

        else:
            # Player 2 free variables
            y = m.addVars(n2, lb=0.0, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="y")
            v = m.addVars(m1, lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="v")

            # Constraint F2.y = f2
            for r_idx, row in enumerate(F2):
                expr = gurobi.LinExpr()
                for c_idx, coeff in row.items():
                    expr.addTerms(coeff, y[c_idx])
                m.addConstr(expr == f2[r_idx], name=f"F2.y[{r_idx}]")

            # Constraint -A.y - F1^T.v >= 0
            for i in range(n1):
                lhs = gurobi.LinExpr()

                for (r_idx, c_idx), value in A.items():
                    if r_idx == i:
                        lhs.addTerms(-value, y[c_idx])

                for r_idx, row in enumerate(F1):
                    coeff = row.get(i, 0.0)
                    if coeff:
                        lhs.addTerms(-coeff, v[r_idx])

                rhs = 0.0
                m.addConstr(lhs >= rhs, name=f"-A.y-F1^T.v[{i}]")

            # Objective max f1^T.v
            obj = gurobi.LinExpr()
            for r_idx in range(m1):
                obj.addTerms(f1[r_idx], v[r_idx])

            m.setObjective(obj, sense=GRB.MAXIMIZE)

        m.optimize()

        if m.Status == GRB.OPTIMAL:
            print(f"Player {player} Objective:", m.getAttr(GRB.Attr.ObjVal))
            print(f"-----------------------------------------------------------------")


def solve_problem_2_2(game):

    S1, F1, f1, S2, F2, f2, A, _, _, _, _, _, _ = build_all_matrices(game)
    n1, n2 = len(S1), len(S2)
    m1, m2 = len(F1), len(F2)

    for player in [1, 2]:

        m = gurobi.Model("game_value_pl" + str(player))

        if player == 1:
```

```
    # Player 1 free variables
    # Reference: https://docs.gurobi.com/projects/optimizer/en/current/reference/python/model.html
    x = m.addVars(n1, vtype=GRB.BINARY, name="x")
    v = m.addVars(m2, lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="v")

    # Constraint F1.x = f1
    # Reference: https://docs.gurobi.com/projects/optimizer/en/current/reference/python/linexpr.html
    for r_idx, row in enumerate(F1):
        expr = gurobi.LinExpr()
        for c_idx, coeff in row.items():
            expr.addTerms(coeff, x[c_idx])
        m.addConstr(expr == f1[r_idx], name=f"F1.x[{r_idx}]")

    # Constraint A^T.x - F2^T.v >= 0
    for j in range(n2):
        lhs = gurobi.LinExpr()

        for (r_idx, c_idx), value in A.items():
            if c_idx == j:
                lhs.addTerms(value, x[r_idx])

        for r_idx, row in enumerate(F2):
            coeff = row.get(j, 0.0)
            if coeff:
                lhs.addTerms(-coeff, v[r_idx])

        rhs = 0.0
        m.addConstr(lhs >= rhs, name=f"A^T.x-F2^T.v[{j}]")

    # Objective max f2^T.v
    obj = gurobi.LinExpr()
    for r_idx in range(m2):
        obj.addTerms(f2[r_idx], v[r_idx])

    m.setObjective(obj, sense=GRB.MAXIMIZE)

else:
    # Player 2 free variables
    y = m.addVars(n2, vtype=GRB.BINARY, name="y")
    v = m.addVars(m1, lb=-GRB.INFINITY, ub=GRB.INFINITY, vtype=GRB.CONTINUOUS, name="v")

    # Constraint F2.y = f2
    for r_idx, row in enumerate(F2):
        expr = gurobi.LinExpr()
        for c_idx, coeff in row.items():
            expr.addTerms(coeff, y[c_idx])
        m.addConstr(expr == f2[r_idx], name=f"F2.y[{r_idx}]")

    # Constraint -A.y - F1^T.v >= 0
    for i in range(n1):
        lhs = gurobi.LinExpr()

        for (r_idx, c_idx), value in A.items():
            if r_idx == i:
                lhs.addTerms(-value, y[c_idx])

        for r_idx, row in enumerate(F1):
            coeff = row.get(i, 0.0)
            if coeff:
                lhs.addTerms(-coeff, v[r_idx])

        rhs = 0.0
        m.addConstr(lhs >= rhs, name=f"-A.y-F1^T.v[{i}]")
```

```
            # Objective max f1^T.v
            obj = gurobi.LinExpr()
            for r_idx in range(m1):
                obj.addTerms(f1[r_idx], v[r_idx])

            m.setObjective(obj, sense=GRB.MAXIMIZE)

        m.optimize()

        if m.Status == GRB.OPTIMAL:
            print(f"Player {player} Objective:", m.getAttr(GRB.Attr.ObjVal))
            print(f"----------------------------------------------------------------------")


def solve_problem_2_3(game):

    S1, F1, f1, S2, F2, f2, A, J1, A1_j, p1_j, J2, A2_j, p2_j = build_all_matrices(game)

    for player in [1, 2]:
        Ji = J1 if player == 1 else J2
        max_k = len(Ji)
        k_values = []
        obj_values = []

        for k in range(0, max_k + 1):
            m = solve_problem_2_3_with_k(player, k, S1, F1, f1, S2, F2, f2, A, J1, A1_j, p1_j, J2, A2_j, p2_j)

            if m.Status == GRB.OPTIMAL:
                print(f"Player {player} Objective with k={k}:", m.getAttr(GRB.Attr.ObjVal))
                k_values.append(k)
                obj_values.append(m.getAttr(GRB.Attr.ObjVal))

            else:
                print(f"Player {player} Objective with k={k} is infeasible.")
                break

        if obj_values:
            plot_objectives(player, k_values, obj_values)
        print(f"----------------------------------------------------------------------")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description='HW2 Problem 2 (Deterministic strategies)')
    parser.add_argument("--game", help="Path to game file", required=True)
    parser.add_argument(
        "--problem", choices=["2.1", "2.2", "2.3"], required=True)

    args = parser.parse_args()
    print("Reading game path %s..." % args.game)

    game = json.load(open(args.game))

    # Convert all sequences from lists to tuples
    # tfsdp = Terminal-Free Sequence-Form Decision Problem
    for tfsdp in [game["decision_problem_pl1"], game["decision_problem_pl2"]]:
        for node in tfsdp:
            if isinstance(node["parent_edge"], list):
                node["parent_edge"] = tuple(node["parent_edge"])
            if "parent_sequence" in node and isinstance(node["parent_sequence"], list):
                node["parent_sequence"] = tuple(node["parent_sequence"])
    for entry in game["utility_pl1"]:
        assert isinstance(entry["sequence_pl1"], list)
        assert isinstance(entry["sequence_pl2"], list)
        entry["sequence_pl1"] = tuple(entry["sequence_pl1"])
```

```
        entry["sequence_pl2"] = tuple(entry["sequence_pl2"])

    print("... done. Running code for Problem", args.problem)

    if args.problem == "2.1":
        solve_problem_2_1(game)
    elif args.problem == "2.2":
        solve_problem_2_2(game)
    else:
        assert args.problem == "2.3"
        solve_problem_2_3(game)
```

References

H. W. Kuhn. A simplified two-person poker. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1 of *Annals of Mathematics Studies, 24*, pages 97–103. Princeton University Press, Princeton, New Jersey, 1950.

Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: opponent modelling in poker. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 550–558, 2005.