15-888 Computational Game Solving (Fall 2021) Released on: Sept. 23, 2021

Homework 1

Student: [*** Your Andrew ID here ***] **Due on: Oct. 7, 2021, beginning of class**

Instructions Please typeset your solution in the tex file provided in the homework zip. Attach a readable printout of your CFR code at the end of the document. Turn in your printed solution at the beginning of class on October 5th. Don't forget to fill in your student ID above :-)

Bonus points This homework contains 3 problems (each split into smaller tasks), for a total of 120 points. The goal of the 20 bonus points is to help you in case you lost some points here and there. The maximum score for the homework will be capped at 100. In other words, your score will be set to the minimum between 100 and the sum of the points you scored on all questions, including the bonus questions.

1 A Regret-Based Proof of the Minmax Theorem (20 points)

Let $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^m$ be convex and compact sets, and $\boldsymbol{A} \in \mathbb{R}^{n \times m}$. The minmax theorem asserts that

$$\max_{\boldsymbol{x} \in \mathcal{X}} \min_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y} = \min_{\boldsymbol{y} \in \mathcal{Y}} \max_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y}.$$

In this problem you will show that the fact that (external) regret minimization algorithms for $\mathcal{X}$ exist is a powerful enough statement to imply the minmax theorem.

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

Problem 1.1 (8 points)**.** Show that

$$\max_{\boldsymbol{x} \in \mathcal{X}} \min_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y} \leq \min_{\boldsymbol{y} \in \mathcal{Y}} \max_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y}. \tag{1}$$

★ Hint: start by arguing that for any $\boldsymbol{y}^* \in \mathcal{Y}$, $\max_{\boldsymbol{x} \in \mathcal{X}} \min_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y} \leq \max_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y}^*$.

Solution. [*** Your solution here ***] □

To show the reverse inequality, we will interpret the bilinear saddle point $\min_{\boldsymbol{y} \in \mathcal{Y}} \max_{\boldsymbol{x} \in \mathcal{X}} \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y}$ as a repeated game. At each time t, we will let a regret minimizer $\mathcal{R}_\mathcal{X}$ pick actions $\boldsymbol{x}^t \in \mathcal{X}$, whereas we will always assume that $\boldsymbol{y}^t \in \mathcal{Y}$ is chosen by the environment to best respond to $\boldsymbol{x}^t$, that is,

$$\boldsymbol{y}^t \in \arg\min_{\boldsymbol{y} \in \mathcal{Y}} (\boldsymbol{x}^t)^\top \boldsymbol{A} \boldsymbol{y}.$$

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

$$\ell_\mathcal{X}^t : \boldsymbol{x} \mapsto \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y}^t = (\boldsymbol{A} \boldsymbol{y}^t)^\top \boldsymbol{x}.$$

By definition of regret minimizer, we will assume that $\mathcal{R}_\mathcal{X}$ guarantees *sublinear regret* in the worst case.

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,

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

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

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

★ Hint: lower bound the maximum over $\boldsymbol{x} \in \mathcal{X}$ by plugging the particular choice $\bar{\boldsymbol{x}}^T$. Then, use the information you have on $\boldsymbol{y}^t$.

Solution. [*** Your solution here ***] □

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

$$\max_{\boldsymbol{x}\in\mathcal{X}} \min_{\boldsymbol{y}\in\mathcal{Y}} \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y} \geq \min_{\boldsymbol{y}\in\mathcal{Y}} \max_{\boldsymbol{x}\in\mathcal{X}} \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y} - \frac{R_{\mathcal{X}}^T}{T}. \tag{3}$$

★ Hint: use the definition of regret $R_{\mathcal{X}}^T$.

Solution. [*** Your solution here ***] □

Problem 1.4 (2 points). Use (3) and (1) and to conclude that

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

Solution. [*** Your solution here ***] □

2 Swap Regret Minimization on Simplex Domains (30 + 10 points)

In this problem you will use the construction of Gordon et al. [2008] seen in Lecture 3 to come up with a regret minimizer for swap regret for single-shot decision making.

We will view swap regret minimization an instance of Φ-regret minimization where the set of transformations Φ is taken to be the set of *all* linear functions that map Δ^n to Δ^n.[1]

We start with a handy characterization of all functions from Δ^n to Δ^n.

[1] In particular, it can be easily shown that Φ contains all functions that map deterministic strategies to deterministic strategies.

Problem 2.1 (7 points). Argue that a linear function $f : \mathbb{R}^n \to \mathbb{R}^n$ maps Δ^n to Δ^n (that is, $f(\Delta^n) \subseteq \Delta^n$) if and only if it can be written in the form

$$f(\boldsymbol{x}) = \boldsymbol{M}\boldsymbol{x},$$

where $\boldsymbol{M}$ is a (column-)stochastic matrix, that is, a nonnegative matrix whose columns all sum to 1.

★ Hint: first of all, any linear function can be written in the form $f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$. Furthermore, if f maps Δ^n to Δ^n, then in particular the vertices $\{\boldsymbol{e}_1, \dots, \boldsymbol{e}_n\}$ of Δ^n get mapped to Δ^n, and so $\boldsymbol{A}\boldsymbol{e}_i \in \Delta^n$. What is $\boldsymbol{A}\boldsymbol{e}_i$?

Solution. [*** Your solution here ***] □

In light of the previous result, we can identify the set of all functions from Δ^n to Δ^n with the set of stochastic matrices, that is,

$$\Phi \equiv \{\boldsymbol{M} \in \mathbb{R}^{n\times n}_{\geq 0} : \boldsymbol{M} \text{ is column-stochastic}\}.$$

Problem 2.2 (10 points). The set of column-stochastic matrices can be described as the concatenation of n independent columns, each of which is a vector in Δ^n. So, a regret minimizer for Φ can be constructed by applying the regret circuit for Cartesian product seen in Lecture 5 [Farina et al., 2019]. Give pseudocode and a formal regret statement as a function of the regret of the individual regret minimizers for the simplexes in the Cartesian product. Is the regret cumulated up to any time T guaranteed to be sublinear in T?

Solution. [*** Your solution here ***] □

Problem 2.3 (7 points). Argue that any column-stochastic matrix $\boldsymbol{M}$ admits a fixed point $\boldsymbol{x} \in \Delta^n$ such that $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{x}$. Then, propose a way to compute such a fixed point.

★ Hint: for the existence, argue that the function $\boldsymbol{x} \mapsto \boldsymbol{M}\boldsymbol{x}$ satisfies the requirements of Brouwer's fixed point theorem.

Solution. [*** Your solution here ***] □

Problem 2.4 (6 points). By combining your solutions to Problems 2.2 and 2.3, give a no-swap-regret algorithm for Δ^n using the construction by Gordon et al. we saw in Lecture 3. What is the bound on the cumulated swap regret?

Solution. [*** Your solution here ***] □

Problem 2.5 (10 bounus points)**.** Compare the algorithm you gave in Problem 2.4 to the algorithm proposed by Blum and Mansour [2007]. Are the algorithms identical or, if not, how do they differ?

Solution. [*** Your solution here ***] □

3 Counterfactual Regret Minimization (50 + 10 points)

In this problem, you will implement the CFR regret minimizer for sequence-form decision problems.

You will run your CFR implementation on three games: rock-paper-superscissors (a simple variant of rock-paper-scissors, where beating paper with scissors gives a payoff of 2 instead of 1) 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, according to the format described 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 $\boldsymbol{A}$ such that the (expected) utility of Player 1 can be written as

$$u_1(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{y},$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are sequence-form strategies for Players 1 and 2 respectively. We represent $\boldsymbol{A}$ 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[2] 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 Learning to best respond

Let Q_1 and Q_2 be the sequence-form strategy polytopes corresponding to the tree-form sequential decision problems faced by Players 1 and 2 respectively. A good smoke test when implementing regret minimization algorithms is to verify that they learn to best respond. In particular, you will verify that your implementation of CFR applied to the decision problem of Player 1 learns a best response against Player 2 when Player 2

[2] A Nash equilibrium of the game is reached when all players play rock with probability $1/2$, paper with probability $1/4$ and superscissors with probability $1/4$. Correspondingly, the game value is 0.

plays the *uniform* strategy, that is, the strategy that at each decision points picks any of the available actions with equal probability.

Let $\boldsymbol{u} \in Q_2$ be the sequence-form representation of the strategy for Player 2 that at each decision point selects each of the available actions with equal probability. When Player 2 plays according to that strategy, the utility vector for Player 1 is given by $\boldsymbol{\ell} \coloneqq \boldsymbol{A}\boldsymbol{u}$, where $\boldsymbol{A}$ is the payoff matrix of the game.

For each of the three games, take your CFR implementation for the decision problem of Player 1, and let it output strategies $\boldsymbol{x}^t \in Q_1$ while giving as feedback at each time t the same utility vector $\boldsymbol{\ell}$. As $T \to \infty$, the average strategy

$$\bar{\boldsymbol{x}}^T \coloneqq \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}^t \in Q_1 \tag{4}$$

will converge to a best response to the uniform strategy $\boldsymbol{u}$, that is,

$$\lim_{T\to\infty} (\bar{\boldsymbol{x}}^T)^\top \boldsymbol{A}\boldsymbol{u} = \max_{\hat{\boldsymbol{x}}\in Q_1} \hat{\boldsymbol{x}}^\top \boldsymbol{A}\boldsymbol{u}.$$

If the above doesn't happen empirically, something is wrong with your implementation.

Problem 3.1 (25 points). In each of the three games, apply your CFR implementation to the tree-form sequential decision problem of Player 1, using as local regret minimizer at each decision point the regret matching algorithm (Lecture 4). At each time t, give as feedback to the algorithm the same utility vector $\boldsymbol{\ell} = \boldsymbol{A}\boldsymbol{u}$, where $\boldsymbol{u} \in Q_2$ is the uniform strategy for Player 2. Run the algorithm for 1000 iterations. After each iteration $T = 1, \dots, 1000$, compute the value of $v^T \coloneqq (\bar{\boldsymbol{x}}^T)^\top \boldsymbol{A}\boldsymbol{u}$ where $\bar{\boldsymbol{x}}^T \in Q_1$ is the average strategy output so far by CFR, as defined in (4).

Plot v^T as a function of T. Empirically, what is the limit you observe v^T is converging to?

★ Hint: represent vectors on $\mathbb{R}^{|\Sigma|}$ (including the sequence-form strategies output by CFR and utility vectors given to CFR) in memory as dictionaries from sequences (tuples `(decision_point_id, action)`) to floats.

★ Hint: in rock-paper-superscissor, v^T should approach the value $1/3$. In Kuhn poker, the value $1/2$. In Leduc poker, the value 2.0875.

Solution. [*** Your solution here. Your solution should include three plots (one for each game), and three values. Don't forget to turn in your implementation. ***] □

3.3 Learning a Nash equilibrium

Now that you are confident that your implementation of CFR is correct, you will use CFR to converge to Nash equilibrium using the self-play idea described in Lecture 3 and recalled next.

The idea behind using regret minimization to converge to Nash equilibrium in a two-player zero-sum game is to use *self play*. We instantiate two regret minimization algorithms, $\mathcal{R}_\mathcal{X}$ and $\mathcal{R}_\mathcal{Y}$, for the domains of the maximization and minimization problem, respectively. At each time t the two regret minimizers output strategies $\boldsymbol{x}^t$ and $\boldsymbol{y}^t$, respectively. Then, they receive as feedback the vectors $\boldsymbol{\ell}^t_\mathcal{X}, \boldsymbol{\ell}^t_\mathcal{Y}$ defined as

$$\boldsymbol{\ell}^t_\mathcal{X} \coloneqq \boldsymbol{A}\boldsymbol{y}^t, \qquad \boldsymbol{\ell}^t_\mathcal{Y} \coloneqq -\boldsymbol{A}^\top \boldsymbol{x}^t, \tag{5}$$

where $\boldsymbol{A}$ is Player 1's payoff matrix.

We summarize the process pictorially in Figure 1.

A well known folk theorem establish that the pair of average strategies produced by the regret minimizers up to any time T converges to a Nash equilibrium, where convergence is measured via the *saddle point gap*

$$0 \le \gamma(\boldsymbol{x}, \boldsymbol{y}) \coloneqq \left(\max_{\hat{\boldsymbol{x}}\in\mathcal{X}}\{\hat{\boldsymbol{x}}^\top \boldsymbol{A}\boldsymbol{y}\} - \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y}\right) + \left(\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{y} - \min_{\hat{\boldsymbol{y}}\in\mathcal{Y}}\{\boldsymbol{x}^\top \boldsymbol{A}\hat{\boldsymbol{y}}\}\right) = \max_{\hat{\boldsymbol{x}}\in\mathcal{X}}\{\hat{\boldsymbol{x}}^\top \boldsymbol{A}\boldsymbol{y}\} - \min_{\hat{\boldsymbol{y}}\in\mathcal{Y}}\{\boldsymbol{x}^\top \boldsymbol{A}\hat{\boldsymbol{y}}\}.$$

A point $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$ has zero saddle point gap if and only if it is a Nash equilibrium of the game.

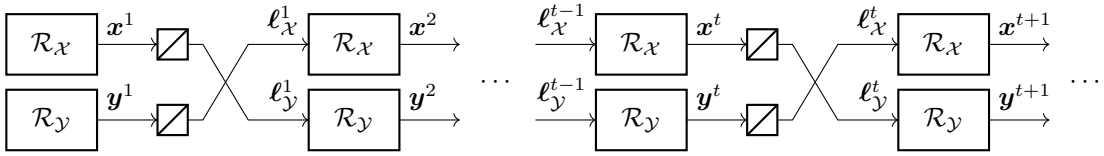

Figure 1: The flow of strategies and utilities in regret minimization for games. The symbol ⊠ denotes computation/construction of the utility vector.

Theorem 1. Consider the self-play setup summarized in Figure 1, where $\mathcal{R}_\mathcal{X}$ and $\mathcal{R}_\mathcal{Y}$ are regret minimizers for the sets $\mathcal{X}$ and $\mathcal{Y}$, respectively. Let $R^T_\mathcal{X}$ and $R^T_\mathcal{Y}$ be the (sublinear) regret cumulated by $\mathcal{R}_\mathcal{X}$ and $\mathcal{R}_\mathcal{Y}$, respectively, up to time T, and let $\bar{\boldsymbol{x}}^T$ and $\bar{\boldsymbol{y}}^T$ denote the average of the strategies produced up to time T, that is,

$$\bar{\boldsymbol{x}}^T \coloneqq \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{x}^t, \qquad \bar{\boldsymbol{y}}^T \coloneqq \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{y}^t. \tag{6}$$

Then, the saddle point gap $\gamma(\bar{\boldsymbol{x}}^T, \bar{\boldsymbol{y}}^T)$ of $(\bar{\boldsymbol{x}}^T, \bar{\boldsymbol{y}}^T)$ satisfies

$$\gamma(\bar{\boldsymbol{x}}^T, \bar{\boldsymbol{y}}^T) \leq \frac{R^T_\mathcal{X} + R^T_\mathcal{Y}}{T} \to 0 \qquad \text{as } T \to \infty.$$

Problem 3.2 (25 points). Let the CFR implementation (using regret matching as the local regret minimizer at each decision point) for Player 1's and Player 2's tree-form sequential decision problems play against each other in self play, as described above.

Plot the saddle point gap and the expected utility (for Player 1) of the average strategies $\gamma(\bar{\boldsymbol{x}}^T, \bar{\boldsymbol{y}}^T)$ as a function of the number of iterations $T = 1, \ldots, 1000$.

★ Hint: represent vectors on $\mathbb{R}^{|\Sigma|}$ (including the sequence-form strategies output by CFR and utility vectors given to CFR) in memory as dictionaries from sequences (tuples `(decision_point_id, action)`) to floats.

★ Hint: to compute the saddle-point gap, feel free to use the function `gap(game, strategy_pl1, strategy_pl2)` provided in the Python stub file.

★ Hint: the saddle point gap shuold be going to zero. The expected utility of the average strategies in rock-paper-superscissor should approach the value 0. In Kuhn poker it should approach -0.055. In Leduc poker it should approach -0.085.

Solution. [*** Your solution here. Your solution should include six plots (two for each game—one for the saddle point gap and one for the utility). Don't forget to turn in your implementation. ***] □

3.4 Bonus: CFR+

To achieve better performance in practice when learning Nash equilibria in two-player zero-sum games, people often make the following modifications to the setup of the previous subsection.

- Instead of regret matching, CFR is set up to use the regret matching plus algorithm (see Lecture 3) at each decision point.
- Instead of using the classical self-play scheme described in Figure 1, people *alternate* the iterates and feedback as described in Figure 2, where the utility vector $\boldsymbol{\ell}^t_\mathcal{X}$ is as defined in (5), whereas

$$\tilde{\boldsymbol{\ell}}^t_\mathcal{Y} \coloneqq -\boldsymbol{A}^\top \boldsymbol{x}^{t+1}.$$

(Note that at the very beginning, $\boldsymbol{x}^1$ does not participate in the computation of any utility vector).

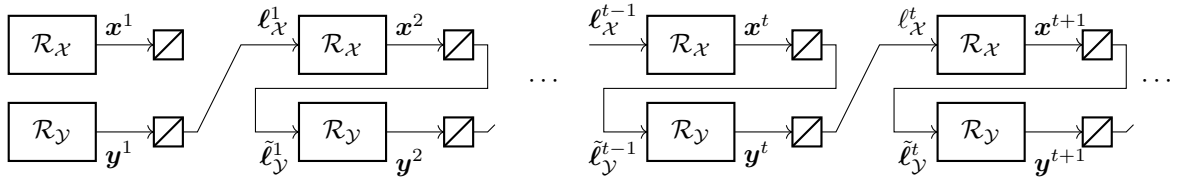

Figure 2: The alternation method for CFR in games. The symbol ⊠ denotes computation/construction of the utility vector.

- Finally, the *linear average* of the strategies, defined as the weighted averages

$$\bar{\bar{\boldsymbol{x}}}^T := \frac{2}{T(T+1)} \sum_{t=1}^{T} t \cdot \boldsymbol{x}^t, \qquad \bar{\bar{\boldsymbol{y}}}^T := \frac{2}{T(T+1)} \sum_{t=1}^{T} t \cdot \boldsymbol{y}^t$$

  is considered instead of the regular averages (6) when computing the saddle point gap.

Collectively, the modified setup we just described is referred to as "running CFR+".

Problem 3.3 (10 bonus points)**.** Modify your implementation of CFR to match the CFR+ self-play setup described above. Run CFR+ for 1000 iterations, plotting the expected utility for Player 1 and the saddle point gap of the linear averages $\gamma(\bar{\bar{\boldsymbol{x}}}^T, \bar{\bar{\boldsymbol{y}}}^T)$ after each iteration T.

★ Hint: represent vectors on $\mathbb{R}^{|\Sigma|}$ (including the sequence-form strategies output by CFR and utility vectors given to CFR) in memory as dictionaries from sequences (tuples `(decision_point_id, action)`) to floats.

★ Hint: to compute the saddle-point gap, feel free to use the function `gap(game, strategy_pl1, strategy_pl2)` provided in the Python stub file.

Solution. [*** Your solution here. Your solution should include six plots (two for each game—one for the saddle point gap and one for the utility). Don't forget to turn in your implementation. ***] □

References

Geoffrey J Gordon, Amy Greenwald, and Casey Marks. No-regret learning in convex games. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 360–367. ACM, 2008.

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

Avrim Blum and Yishay Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8(6), 2007.

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.