CS639: Algorithmic Game Theory & Learning

University of Wisconsin–Madison, Spring 2026 *Instructor: Kirthevasan Kandasamy*

Homework 4. Due 03/27/2026, 11.59 pm

Instructions:

1. Homework is due on Canvas by 11:59 pm on the due date. Please plan to submit well before the deadline. Refer to the course website for policies on late submission.
2. Homework must be typeset using appropriate software, such as LaTeX. Handwritten and scanned submissions will **not** be accepted.
3. Your solutions will be evaluated on correctness, clarity, and conciseness.
4. Unless otherwise specified, you may use any result we have already discussed in class. Clearly state which result you are using.
5. If you use any external references, please cite them in your submission.
6. **Collaboration:** You may collaborate in groups of size up to 3. If you collaborate, please indicate your collaborators at the beginning of the homework. Even if you collaborate, *you must write the solution in your own words*.

1 Mechanism design in the real world

Describe a real-world system where the goals of the participants are misaligned with that of a social planner, leading to strategic/manipulative behavior from the participants, and consequently socially less desirable outcomes. Suggest at least one alternative design which will completely or partially address these considerations. Your solution should:

1. **[2 pts]** Describe the problem.
2. **[3 pts]** Intuitively describe the reasons for strategic behavior along with empirical or anecdotal evidence that participants are doing so.
3. **[2 pts]** Describe how it leads to less desirable social outcomes.
4. **[3 pts]** Explain how your solution will mitigate or completely eliminate strategic considerations.

Limit your answer to at most 500 words. Please see an example solution below. You will need to come up with a different example.

1. ***Problem: strategic voting.*** *Strategic voting is when voters vote for someone other than their preferred candidate due to fears of "wasting" their vote on a candidate, who they prefer, but is unlikely to win.*
2. *Strategic voting is common in first past the post (FPP) elections where the candidate with the most number of votes win. Suppose there are three candidates a, b, c and a voter's preference is $a \succ b \succ c$. However, if the voter believes that a has very little support among the broader population, they might choose to vote for b to reduce the chances of c winning the election. Strategic voting has been documented in many countries. Examples include the US presidential elections, where voters choose to vote for either Democratic or Republican candidates over third-party candidates, and in UK parliamentary elections where voters vote for the big parties (e.g Conservatives, Labour, SNP) over the smaller parties.*
3. *This is undesirable for several reasons. For instance, voters who prefer the smaller parties may never be represented in government. Moreover, even if there is in fact large support for the smaller parties, this may never be revealed if everyone simply keeps voting for the larger parties. As a result, the government may not necessarily reflect the electorate.*
4. ***Alternative design 1: ranked-choice voting.*** *In ranked choice voting, voters submit a ranked list of candidates instead of choosing a single candidate. Vote counting happens through a series of rounds. In the first round, we first count everyone's first choice vote. We then eliminate the candidate with the fewest votes and count the next choice of these voters. We proceed in this fashion until two candidates are remaining and the candidate with the highest vote share wins. This reduces considerations about strategic voting. For instance, in the example above, the voter will know that if a does not get many votes, their vote will count towards b and not be wasted.*

   Alternative design 2: proportional representation. *We may also consider moving to a system of government that uses proportional representation instead of FPP. Here, instead of selecting a single candidate, we may elect multiple candidates from multiple parties from the same electorate, where the number of candidates assigned to a party is proportional to the number of votes the party received in the electorate. This mitigates considerations about strategic voting and also ensures representation from minority parties.*

N.B: While ranked choice voting reduces the risk of strategic voting, it does not eliminate it entirely. A negative (and rather depressing) result in social choice theory, Arrow's Impossibility Theorem, states that the only voting system for electing a single winner that is immune to strategic manipulation is a dictatorship, where a single voter determines the outcome. One way to implement this is to designate an arbitrary person as a permanent dictator who selects their favorite candidate (or themselves). A fairer, yet still dictatorial, approach is *random dictatorship*: randomly selecting a voter after all votes are submitted and electing that voter's preferred candidate. Since no one knows in advance who will be chosen, strategic manipulation is effectively neutralized. See KP Chapter 13 for more details.

2 No-swap-regret dynamics converges to a CE

In an n player game, let player i's action space be $\mathcal{A}_i$. Denote $\mathcal{A} = \bigtimes_{i=1}^{n} \mathcal{A}_i$ and $\mathcal{A}_{-i} = \bigtimes_{j \neq i} \mathcal{A}_j$. Let $u_i : \mathcal{A} \to [-1/2, 1/2]$ be the (bounded) utility function of player i, where $u_i(a)$ is her utility under action profile $a \in \mathcal{A}$.

In this question, we will show that no-swap-regret dynamics converge to a correlated equilibrium (CE). To do so, we will first introduce a specific definition for an approximation of a CE (one can define other ways to approximate CEs, but this one is useful for no-swap-regret dynamics). To motivate this notion of approximation, we will first develop an equivalent definition of a CE. Recall that a joint distribution $s \in \Delta(\mathcal{A})$ is a CE if, for all players i, we have

$$\mathbb{E}_{a\sim s}\left[u_i(a_i', a_{-i})|a_i = a_i'\right] \geq \mathbb{E}_{a\sim s}\left[u_i(a_i'', a_{-i})|a_i = a_i'\right], \quad \text{for all } a_i', a_i'' \in \mathcal{A}_i. \tag{1}$$

1. **[9 pts]** *(Equivalent definition of a CE)* A function $\sigma_i : \mathcal{A}_i \to \mathcal{A}_i$ which maps each action of player i to another action is called a *swap function* for player i. Show that s is a CE if and only if for all players i, we have

$$\mathbb{E}_{a\sim s}\left[u_i(a)\right] \geq \mathbb{E}_{a\sim s}\left[u_i(\sigma_i(a_i), a_{-i})\right], \quad \text{for all swap functions } \sigma_i. \tag{2}$$

   Hint: To show (2) $\implies$ (1), consider a swap function σ_i of the following form for a given a_i', a_i'':

$$\sigma_i(k) = \begin{cases} a_i'' & \text{if } k = a_i', \\ k & \text{otherwise.} \end{cases}$$

The result from part 1 motivates the following definition for an approximate CE. We say that a joint distribution s is an ϵ-approximate CE if

$$\mathbb{E}_{a\sim s}\left[u_i(a)\right] \geq \mathbb{E}_{a\sim s}\left[u_i(\sigma_i(a_i), a_{-i})\right] - \epsilon, \quad \text{for all swap functions } \sigma_i. \tag{3}$$

Notation. To reduce notational clutter, we will adopt the following shorthands. For a joint distribution $s \in \Delta(\mathcal{A})$ let $u_i(s) \triangleq \mathbb{E}_{a\sim s}[u_i(a)]$ denote the expected utility of agent i. Next, for an action $k \in \mathcal{A}_i$ and joint distribution $s_{-i} \in \Delta(\mathcal{A}_{-i})$ overy everyone's actions except i's, let $u(k, s_{-i}) \triangleq \mathbb{E}_{a_{-i}\sim s_{-i}}[u_i(k, a_{-i})]$ denote the expected utility of agent i when she follows action k and others' actions are sampled according to s_{-i}.

No-swap-regret dynamics. Now, suppose that the game is repeated over T rounds. On each round, all players *simultaneously* choose mixed strategies. On round t, player i chooses a mixed strategy $p_{i,t} \in \Delta(\mathcal{A}_i)$ via some policy π_i. Then player i's action $A_{i,t}$ is sampled from $p_{i,t}$. At the end of round t, player i observes $\left\{u_i(k, p_{-i,t})\right\}_{k\in\mathcal{A}_i}$, where $p_{-i,t} = \bigtimes_{j\neq i} p_{j,t}$ is the product distribution of others' actions on round t. That is, she observes the expected utility she would have received for each possible action, having fixed the mixed strategies of the other players. The loss of player i on round t for choosing action $k \in \mathcal{A}_i$ is $\ell_{i,t}(k) = \frac{1}{2} - u_i(k, p_{-i,t})$. Let $\sigma_i : \mathcal{A}_i \to \mathcal{A}_i$ be a given swap function for player i. Define the swap regret $R_{i,T}^{\text{sw}}$ of player i with respect to this swap function as,

$$R_{i,T}^{\text{sw}}(\sigma_i) = \mathbb{E}\left[\sum_{t=1}^{T} \ell_{i,t}(A_{i,t}) - \sum_{t=1}^{T} \ell_{i,t}(\sigma_i(A_{i,t}))\right].$$

2. **[5 pts]** *(Convergence to CE.)* Let $\epsilon_T = \max_{i\in[n]} \max_{\sigma_i} \frac{1}{T} R_{i,T}^{\text{sw}}(\sigma_i)$ be the maximum time-averaged swap regret of any player $i \in [n]$ for any swap function $\sigma_i : \mathcal{A}_i \to \mathcal{A}_i$. Let $\overline{p} = \frac{1}{T}\sum_{t=1}^{T} p_t$ be the time-averaged distribution of all players[1]. Show that $\overline{p}$ is an ϵ_T–approximate CE as defined in (3).

   Hint: As each $p_t = \bigtimes_{j=1}^{n} p_{j,t}$ is a product distribution, we can write, $\mathbb{E}_{a\sim\overline{p}}\left[u_i(a)\right]$ as shown below. (We used a similar observation when proving that no-regret dynamics converges to a CCE in class.) We have,

$$\mathbb{E}_{a\sim\overline{p}}\left[u_i(a)\right] = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{a\sim p_t}\left[u_i(a)\right] = \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{A_{i,t}\sim p_{i,t}}\left[u_i(A_{i,t}, p_{-i,t})\right].$$

3. **[6 pts]** *(Algorithms for approximating a CE.) Based on your solution to part 2*, describe a procedure to compute an approximate CE in any finite n-player general sum game. State how many iterations are required (big-O notation is sufficient), to find an ϵ–approximate CE.

   N.B. Your solution must be based on *no-swap-regret* dynamics derived in part 2. For instance, stating that we can compute an exact CE via an LP is *not* an acceptable solution.

[1] Note that this may not be a product distribution and is not equal to $\overline{p}_1 \times \cdots \times \overline{p}_n$ where $\overline{p}_i = \frac{1}{T}\sum_{t=1}^{T} p_{i,t}$.

3 Computing NE in two player zero sum games

In this problem, you will compute NE for two player zero sum games via linear programs and no-regret dynamics (NRD). We will assume that player 1 has m actions and player 2 has n actions. Let $Q \in \mathbb{R}^{m \times n}$ denote the game's payoff matrix where $u_1(i,j) = -u_2(i,j) = Q_{i,j}$.

1. **[2 pts]** *(Nash equilibrium via LP)* Write LPs to compute a Nash equilibrium in a two player zero sum game with payoff matrix $Q \in \mathbb{R}^{m \times n}$.

2. **[3 pts]** *(Computing losses in NRD)* Suppose that the game is repeated for multiple rounds, and on round t player 1 chooses x_t and player two chooses y_t. Write out the losses $\ell_{1,t} \in [0,1]^m$ and $\ell_{2,t} \in [0,1]^n$ which can be passed onto a policy (e.g., Hedge) so that each player can achieve no regret.

3. **[15 pts]** *(Implementation)* See the Python starter code provided with this homework. Implement the following functions/classes in the file `solve_zero_sum_games.py`.

   (a) `compute_tpzsg_ne_via_lp`. Using your solution to part 1, write LPs to compute a Nash equilibrium of a two-player zero-sum game. The input to this function is a matrix `Q`, which denotes the payoff matrix Q.

   (b) `compute_tpzsg_approx_ne_via_nrd`. Using your solution to part 2, implement no-regret dynamics to compute an ϵ-approximate Nash equilibrium of a zero-sum game, *using Hedge as the no-regret policy for both agents*. The inputs to this function are a matrix `Q` and a scale `eps`, which respectively denote the payoff matrix Q and the approximation parameter ϵ.

   Run NRD for `num_iters` iterations with learning rates `eta_1` and `eta_2` for Player 1 and Player 2 respectively. Using these values is necessary to obtain the exact outputs produced by our implementation.

   To implement NRD, you will also need to implement Hedge. You are encouraged to follow the template in the provided Python class `Hedge`. You may instantiate `Hedge` objects with appropriate learning rates when implementing NRD; at the beginning of each round invoke `get_pt` to obtain the action probabilities for that round; at the end of the round, invoke `update` to update the cumulative losses (the starter code shows you how to do this). If you choose to follow this template, implement the following two methods under the `Hedge` class.

   (c) `update`. Given the input `losses_for_curr_round`, which denotes the losses ℓ_t in round t, update the cumulative losses for each action via the attribute `self.cumulative_losses`.

   (d) `get_pt`. Compute the probability distribution for the current round based on the cumulative losses so far. The learning rate is stored in the attribute `self.eta`.

   If you choose not to follow this template, please still use the same number of iterations and learning rate(s) as specified in part 3b so that your output matches our reference implementation.

   Please follow these instructions carefully:

   - Execute `python main.py` to execute the code and generate the required outputs. The starter code is set up to produce these outputs automatically. **Please print the outputs** generated by the starter code **for all four games provided**. Your grade will be based on the outputs your program produces.
   - You may use any LP solver of your choice (our implementation uses SciPy's `optimize.linprog` module).
   - If you choose to use a language other than Python, you are responsible for generating all instances exactly as in the starter code, and writing any necessary boilerplate to reproduce the required outputs.
   - Below, we display the output for the first game. If your output does not match ours, first double-check your implementation with your peers. You may post your output for the first game on Piazza if it differs from the reference output.

```
Game 1: random_zero_sum_game_3_4 (m=3, n=4)
    - Computing NE via LP
        - Player 1 NE strategy: [0.8665998 0.        0.1334002]
        - Player 2 NE strategy: [0.47041123 0.52958877 0.         0.        ]
        - Time taken: 0.0049s
    - Computing an 0.005-approxiate NE via no-regret dynamics
```

```
- Player 1 approx NE strategy: [0.863852187752903,
       0.0020398333613053043, 0.13410797888857725]
- Player 2 approx NE strategy: [0.46951323796193506, 0.5275275965081019,
                          0.0013688907120900109, 0.0015902748178785017]
- Approximation to NE: 0.0010924908338793143.  Is smaller than 0.005? Yes.
- Time taken: 7.3839s
```

4 On the Gale-Shapley algorithm

1. **[4 pts]** *(Execution of Gale-Shapley.)* Let $\mathcal{F} = \{a, b, c\}$ be a set of firms and let $\mathcal{W} = \{x, y, z\}$ be a set of workers. Consider the following preferences.

$$p_a = y \succ_a x \succ_a z, \qquad p_b = x \succ_b y \succ_b z, \qquad p_c = x \succ_c z \succ_c y.$$
$$p_x = a \succ_x b \succ_x c, \qquad p_y = b \succ_y a \succ_y c, \qquad p_z = c \succ_z a \succ_z b.$$

   Execute the firm-proposing version of the Gale-Shapley algorithm and compute the matching returned. Note that the order in which you pick the firm to propose does not matter as it will all eventually terminate at the same matching (Why?). You may use the following template for your solution.

Round	Proposal in current round	Matches after proposal	Unmatched
0			$a, b, c,$ x, y, z
1	$a \to y$	$a \leftrightarrow y$	$b, c,$ x, z
$\vdots$	$\vdots$	$\vdots$	$\vdots$

2. *(Properties of Gale-Shapley.)* In this question, you will prove some properties about the Gale-Shapley algorithm we discussed, but did not prove, in class.

   (a) **[6 pts]** We say that a firm f is attainable for a worker w if there exists a stable matching μ with $\mu(f) = w$. We can similarly define attainability for workers. Consider the firm-proposing version of the Gale-Shapley algorithm. Prove that each worker is matched to her least preferred attainable firm.

   Hint. You may use a proof by contradiction. You will need to use the fact that in the Gale-Shapley algorithm, each firm is matched to its most preferred attainable worker.

   (b) **[4 pts]** Show, via a counter example, that the firm-proposing version of the Gale-Shapley algorithm is not incentive-compatible for the workers. That is, a worker can get a better match by misreporting her preferences (even when all firms and all other workers are reporting truthfully).

   Hint: To construct this counter-example, you will need to consider at least 3 firms and workers. You may build on the example in part 1.

5 Matching by compatibility

A set of n firms and n workers wish to be matched to each other. Suppose there exists a matrix $A \in \mathbb{R}_+^{n \times n}$, where $A_{i,j}$ represents the compatibility between firm i and worker j. We will assume that each entry of this matrix is unique, i.e., $A_{i,j} \neq A_{i',j'}$ for all $(i, j) \neq (i', j')$. Each firm prefers a more compatible worker to a less compatible one, and vice versa for workers. That is,

$$A_{i,j} > A_{i,j'} \iff j \succ_i j' \quad \text{for all firms } i.$$
$$A_{i,j} > A_{i',j} \iff i \succ_j i' \quad \text{for all workers } j.$$

1. *(Maximum compatible matching.)* Consider the matching which maximizes the compatibility, i.e $\sum_{i=1}^{n} A_{i,\mu(i)}$, among all possible matchings μ. This is the maximum weight bipartite matching between firms and workers, where the weights are given by A. State if the following statements are true or false. If true, outline a proof/explanation and if false, provide a counter-example.

   (a) **[2 pts]** *(Uniqueness)* A maximum compatible matching exists uniquely.

(b) **[3 pts]** *(Stability)* A maximum compatible matching is always stable.

(c) **[3 pts]** *(Incentive-compatibility)* Suppose that we (mechanism designer) did not know the matrix A a priori and have to rely on the firms to report the compatibility scores. In particular, firm i will report n values $\{B_{i,j}\}_{j\in[n]}$ (not necessarily truthfully), where $B_{i,j}$ represents $A_{i,j}$. We then return a maximum compatible matching. In this setting, each firm i is incentivized to report the compatibility values truthfully (i.e., $B_{i,j} = A_{i,j}$ for all j) regardless of the other firms' reports.

2. *(Stable matching.)* Suppose that we instead wish to find a stable matching between the firms and the workers. State if the following statements are true or false. If true, outline a proof/explanation and if false, provide a counter-example.

(a) **[6 pts]** *(Uniqueness)* Under the above model for preferences, a stable matching exists uniquely.

(b) **[4 pts]** *(Incentive-compatibility)* Suppose that we need to rely on the firms to report the compatibility scores, similar to part 1c. In this setting, each firm is incentivized to report the compatibility values truthfully regardless of the other firms' reports.