ML-Decision Project

Bar Amrani 204117071
Amir Steiner 302342498

March 2025

Abstract

This project presents a practical implementation and theoretical exploration of the research detailed in "Competing for Shareable Arms in Multi-Player Multi-Armed Bandits." The paper introduces a novel framework in which multiple self-interested players compete over arms that yield rewards shared among all users, capturing realistic settings such as spectrum-sharing and content recommendation. Building on a rigorous game-theoretic model, the authors prove the existence of a unique pure Nash equilibrium and design the SMAA (Selfish MPMAB with Averaging Allocation) algorithm. SMAA leverages empirical estimation, KL-UCB-based exploration, and block-based coordination to achieve logarithmic regret and rapid convergence to equilibrium, while ensuring stability against unilateral deviations. In our work, we summarize the key theoretical results, provide a detailed implementation of the SMAA algorithm, and validate its performance via extensive simulations. Furthermore, we propose an extension that incorporates time-varying rewards and heterogeneous player preferences, thereby broadening the algorithm's applicability to dynamic real-world environments. Overall, our study demonstrates the efficiency, robustness, and optimality of SMAA in decentralized multi-agent settings.

1 Summary of "Competing for Shareable Arms in Multi-Player Multi-Armed Bandits"

1.1 Introduction and Motivation

In multi-player multi-armed bandits (MPMAB), multiple self-interested players interact with a set of arms (or actions). Traditional collision-based models assume that if more than one player chooses the same arm, their individual rewards are heavily diminished or zero (e.g., (8)). However, many real-world resources are *shareable*, meaning that even if multiple players select the same arm, their individual payoffs may be reduced but remain nonzero. Examples include:

- **Spectrum-sharing** in cognitive radio networks, where a channel can be partially occupied by multiple users simultaneously.
- **Recommender systems** such as YouTube or TikTok, which serve content to many users in parallel. Selecting a "popular" arm does not necessarily yield zero reward for each user, because the content can be shared or recommended to multiple people at once.

1.1.1 Main Claim

This study proposes a new mechanism for handling strategic interactions in an MPMAB setting with shareable resources. Specifically, it argues that the *Selfish MPMAB with Averaging Allocation* (SMAA) algorithm:

1. Converges to a Nash equilibrium in which players' allocations approximate the arms' relative merits,

2. Achieves an individual regret of order $O(\log T)$ (asymptotically optimal),

3. Exhibits robustness against unilateral strategic deviations.

1.2 Literature Review & Context

Multi-player multi-armed bandits (MPMAB) problems have drawn increasing attention as they extend the classical single-agent bandit framework to settings where multiple players interact with limited resources. Early research primarily examined cooperative multi-agent scenarios in which collisions yield zero reward, focusing on minimizing total group regret (Besson & Kaufmann, 2018; Rosenski, Shamir, & Szlak, 2016). More recent work explored competitive environments in which players act selfishly and collisions typically benefit only one participant (Jagadeesan et al., 2021; Liu, Mania, & Jordan, 2020). However, these traditional models do not fully reflect real-world applications—like cognitive radio networks (Bande, Magesh, & Veeravalli, 2021; Riahi & Riahi, 2019) and algorithmic content platforms (Ben-Porat, Rosenberg, & Tennenholtz, 2020)—where resources are often shareable rather than strictly exclusive.

The paper addresses this gap by proposing a novel averaging allocation mechanism. Rather than losing the reward entirely upon collision or handing it all to a single winner, multiple players selecting the same arm split its payoff. This framework—building on ideas from capacity-based (Bande et al., 2021) and proportional-sharing models (Shi & Shen, 2021)—more accurately captures how agents in shared environments (e.g., spectrum access, online content feeds) might divide and learn resource values. In addition, the analysis provides a game-theoretic perspective, showing how equilibria (Roughgarden, 2005) form under selfish incentives and proving convergence to stable outcomes. By demonstrating unique pure-strategy Nash equilibria, bounding the Price of Anarchy, and introducing the Selfish MPMAB with Averaging Allocation (SMAA) algorithm, the work advances our theoretical understanding of strategic learning dynamics in bandit problems with shareable resources and offers practical insights for resource allocation in modern multi-user systems.

1.3 Unified Model, Assumptions, and Equilibrium Analysis

We consider a decentralized multi-player multi-armed bandit (MPMAB) setting with K arms indexed by $[K] = \{1, 2, \ldots, K\}$, N players, and T rounds. At each round $t \in [T]$, each arm k produces an i.i.d. reward $X_k(t)$ drawn from a distribution ξ_k with expectation $\mu_k > 0$ (supported on $[0, 1]$). Each player j selects an arm $\pi_j(t)$, and the number of players choosing arm k is defined as

$$M_k(t) \;=\; \sum_{j=1}^{N} \mathbf{1}\{\pi_j(t) = k\}. \tag{1}$$

Under the *averaging allocation* mechanism, if multiple players choose the same arm, they share its reward equally. Concretely, assuming that each player is assigned a random weight $w_j(t)$ drawn i.i.d. from a distribution Γ (and normalized among the players selecting arm k), the expected reward of player j given $\pi_j(t) = k$ is

$$\mathbb{E}[R_j(t) \mid \pi_j(t) = k] \;=\; \frac{X_k(t)}{M_k(t)}.$$

Thus, as more players crowd onto an arm, the individual payoff decreases (in expectation).

Assumption 1 (No-Ties). *For any two distinct arms $k_1 \neq k_2$ and for any integers $n_1, n_2 \in \{1, \dots, N\}$, we assume that*

$$\frac{\mu_{k_1}}{n_1} \neq \frac{\mu_{k_2}}{n_2}.$$

This assumption holds with probability 1 when the μ_k values are drawn from a continuous distribution.

1.3.1 One-Shot Game: Equilibrium and Price of Anarchy

Assume that, for a one-shot game, the expected rewards $\{\mu_k\}$ are common knowledge. In this setting, the utility of a player j who chooses arm k is

$$U_j(k) = \frac{\mu_k}{m_k},$$

where m_k is the number of players selecting arm k.

1.3.2 Pure Nash Equilibrium

Under Assumption 1, one can prove that there exists a unique pure Nash equilibrium in terms of the allocation of players across arms. Specifically, there is a unique integer vector

$$m^* = \big(m_1^*, m_2^*, \dots, m_K^*\big), \tag{2}$$

satisfying $\sum_{k=1}^K m_k^* = N$ such that

$$m_k^* = \left\lfloor \frac{\mu_k}{z^*} \right\rfloor, \quad z^* = \sup\Big\{z > 0 : \sum_{k=1}^K \left\lfloor \frac{\mu_k}{z} \right\rfloor \geq N\Big\}.$$

A pure-strategy profile $s = (s_1, \dots, s_N)$ is a Nash equilibrium if and only if exactly m_k^* players choose arm k for each k.

Theorem 1 (Pure Nash Equilibrium). *There exists a unique allocation m^* (see Equation* (2)*) under Assumption 1, and any pure-strategy profile that assigns exactly m_k^* players to each arm k is a Nash equilibrium.*

Remark 1 (Implementation). *Determining m^* requires knowledge of the μ_k values. In practice, with random tie-breaking under mild conditions, the threshold z^* is well-defined, leading to a unique division of players across arms.*

1.3.3 Price of Anarchy (PoA)

Definition 1 (Price of Anarchy). *We define the Price of Anarchy (PoA) as*

$$\text{PoA} = \frac{\textit{Maximum Social Welfare}}{\textit{Social Welfare at Equilibrium}}.$$

The PoA measures the efficiency loss due to strategic behavior by comparing the optimal social welfare with the welfare achieved at equilibrium (see (9)).

1.4 Online Learning via SMAA Algorithm

When the true expected rewards $\{\mu_k\}$ are unknown, each player must both learn these values and compete for shareable rewards. The paper proposes the **SMAA** (Selfish MPMAB with Averaging Allocation) algorithm, which operates in blocks of N rounds. In summary, SMAA enables each of the N players to estimate arm qualities and coordinate their actions to approximate the Nash equilibrium allocation m^* (as defined in Theorem 1). The algorithm consists of three main components:

- **Initialization:** Every arm is pulled at least once over a total of $K' = N \cdot \lceil K/N \rceil$ rounds, providing initial estimates of the arms' expected rewards.
- **Rank-Based Assignment and Equilibrium Estimation:** Each player computes empirical means $\hat{\mu}_k(t)$ and, using these, approximates the equilibrium allocation by substituting into the equilibrium formula. This determines the desired number of players per arm.
- **Exploration via KL-UCB:** To avoid premature convergence to suboptimal allocations, players compute KL-UCB indices that encourage exploration by quantifying the uncertainty in their estimates.

The performance of SMAA is evaluated by two key metrics:

1. **Regret:** The cumulative loss incurred by not obtaining the best possible per-round reward.
2. **Non-Equilibrium Rounds:** The number of rounds during which the actual allocation deviates from the equilibrium allocation m^*.

Note: The detailed pseudocode for the SMAA algorithm is provided in Appendix 4.

Key Theoretical Results and Practical Implications

1.4.1 Regret Upper Bound

Definition 2 (Regret)**.** *For each player j, the cumulative regret over T rounds is defined as*

$$\mathrm{Reg}_j(T) \;=\; \sum_{t=1}^{T}\Big(\max_{k\in[K]} \frac{\mu_k}{M_k(t)} - R_j(t)\Big),$$

where $M_k(t)$ is given in (1) *and $R_j(t)$ is the reward obtained by player j.*

Theorem 2 (Regret Upper Bound)**.** *Under Assumption 1 and for parameters satisfying* $0 < \delta < \delta_0/2$ *(with δ_0 being the minimum gap among the set $\{\mu_k/n : 1 \le k \le K,\ 1 \le n \le N\} \cup \{0\}$), if all players follow the SMAA algorithm then*

$$\mathbb{E}[\mathrm{Reg}_j(T)] \;\le\; \sum_{k\notin M^*} \frac{(z^* - \mu_k)\,\big(\log T + 4\log\log T\big)}{\mathrm{kl}\big(\mu_k+\delta,\; z^*-\delta\big)} \;+\; \mathcal{O}\Big(N^3 K\big(\delta^{-2}+K\big)\Big),$$

where

$$z^* \;=\; \sup\Big\{z>0 : \sum_{k=1}^{K} \Big\lfloor \frac{\mu_k}{z} \Big\rfloor \ge N\Big\},$$

and $\mathrm{kl}(\cdot,\cdot)$ *is the Kullback–Leibler divergence for Bernoulli distributions.*

This shows that the cumulative regret grows only logarithmically with the time horizon T. In practice, this means that over long durations, the loss due to suboptimal arm selections remains minimal.

1.4.2 Non-Equilibrium Rounds

Definition 3 (Non-Equilibrium Rounds). *Let*

$$\mathrm{NonEqu}(T) \;=\; \sum_{t=1}^{T} \mathbf{1}\Big\{\exists\, k \text{ such that } M_k(t) \neq m_k^*\Big\},$$

i.e., the count of how many rounds deviate from the theoretical Nash equilibrium allocation m^*.

Theorem 3 (Non-Equilibrium Rounds). *Under the same conditions as Theorem 2,*

$$\mathbb{E}[\mathrm{NonEqu}(T)] \;\le\; N \sum_{k \notin M^*} \frac{\log T + 4 \log\log T}{\mathrm{kl}\big(\mu_k + \delta,\; z^* - \delta\big)} \;+\; \mathcal{O}\Big(N^3 K \big(\delta^{-2} + K\big)\Big).$$

Hence, the system converges quickly to a stable allocation, ensuring consistent performance.

1.4.3 (β, ε)-Stability and ε-Nash Equilibrium

Definition 4 (ε-Nash Equilibrium). *A policy profile is an ε-Nash equilibrium if no single player can unilaterally improve their payoff by more than ε.*

Definition 5 ((β, ε)-stability). *A policy profile is (β, ε)-stable if, whenever a unilateral deviation reduces another player's reward by an amount $u > 0$, the deviator's own reward also decreases by at least $\beta\, u - \varepsilon$.*

Theorem 4 (Stability and ε-Nash). *Under Assumption 1 and the same parameter settings, the SMAA profile is an ε-Nash equilibrium and is $(\beta, \varepsilon + \beta\gamma)$-stable, where*

$$\beta \;=\; \frac{\delta_0}{z^*}, \quad \varepsilon \;=\; \mathcal{O}(\log T), \quad \gamma \;=\; \mathcal{O}(\log T).$$

This result ensures that unilateral deviations yield negligible benefits (at most ε) and impose comparable costs, making the system robust against selfish behavior.

1.4.4 Regret Lower Bound and Consistency

Definition 6 (Consistency). *A policy profile $A = (A_1, \ldots, A_N)$ is called* consistent *if, for every instance $\{\xi_k\}_{k=1}^{K} \in \mathcal{I}_{N,K}^{\mathrm{ber}}$ (Bernoulli distributions satisfying Assumption 1) with means $\mu_k = \mathbb{E}[\xi_k]$, and for each $k \in M^*$ and any player j, the difference*

$$\frac{m_k^* T}{N} - \mathbb{E}[\tau_{j,k}(T)]$$

grows slower than any positive power of T (i.e., $o(T^\alpha)$ for all $\alpha > 0$), where $\tau_{j,k}(T)$ is the number of times player j selects arm k up to time T.

Theorem 5 (Regret Lower Bound). *If a policy profile A is consistent and uses only reward observations (no side information), then for any instance in $\mathcal{I}_{N,K}^{\mathrm{ber}}$ and for each arm $k \in M^*$,*

$$\liminf_{T\to\infty} \frac{\mathbb{E}[\tau_{j,k}(T)]}{\log T} \;\ge\; \frac{1}{\mathrm{kl}(\mu_k, z^*)},$$

and

$$\liminf_{T\to\infty} \frac{\mathbb{E}[\mathrm{Reg}_j(T)]}{\log T} \;\ge\; \sum_{k \notin M^*} \frac{z^* - \mu_k}{\mathrm{kl}(\mu_k, z^*)}.$$

Thus, no consistent policy can achieve regret lower than $\Omega(\log T)$. Since SMAA achieves an $O(\log T)$ regret upper bound, it is asymptotically optimal.

1.4.5 Price of Anarchy (PoA) and Social Welfare Efficiency

Recalling Definition 1, the PoA compares the maximum achievable social welfare with that attained at equilibrium. In this shareable-arms MPMAB game, the following proposition bounds the PoA.

Proposition 1 (Price of Anarchy Bound)**.** *Under Definition 1, we have*

$$1 \;\leq\; \mathrm{PoA} \;\leq\; \frac{N + \min\{K, N\} - 1}{N}.$$

In certain cases, the equilibrium is fully efficient ($\mathrm{PoA} = 1$*), while in other scenarios the PoA may approach this upper bound.*

Hence, even under strategic competition, the degradation in total welfare is capped by a known factor.

1.5 Experimental Validation and Practical Implications

Experimental results confirm that SMAA achieves logarithmic regret and rapid convergence to the Nash equilibrium. Simulations using Beta and Bernoulli rewards show that non-equilibrium rounds decline quickly, ensuring minimal performance loss over time. Moreover, the robustness guarantees—such as (β, ε)-stability and bounded Price of Anarchy—demonstrate that individual deviations have negligible impact, making SMAA a scalable and effective solution for decentralized applications.

1.6 Extensions, Discussion, and Conclusion

The SMAA algorithm can be adapted to unknown numbers of players or ranks by combining ideas from the "Musical Chairs" approach (8), still achieving $O(\log T)$ regret. This is valuable for resource allocation problems in wireless networks or recommendation platforms, where agents do not coordinate. Because μ_k can be continuous, the no-ties assumption holds with probability 1 in practice.

1.7 Final Remarks

Under Assumption 1, a unique PNE exists and SMAA converges to it with $O(\log T)$ regret and few non-equilibrium rounds. It also remains stable if players act maliciously. No consistent policy can improve on $O(\log T)$ regret, so SMAA is asymptotically optimal, illustrating that stable equilibria can be learned online in a multi-agent shareable-arms setting.

2 Implementation

Our code available at: https://github.com/AmraniBar/MLIDProject.git

Structure
Our implementation is organized into several Python modules, each serving a specific function:

2.1 `environment.py`

- **Purpose:** Creates a multi-player multi-armed bandit environment where arms are shareable.
- **Implementation:**
  - Initializes Beta or Bernoulli arms.
  - Implements a `pull(...)` method that divides each arm's reward among players who select it, mirroring the "averaging allocation" from the paper.

2.2 `equilibrium.py`

- **Purpose:** Computes the Nash Equilibrium (m^*) as defined in the paper.
- **Implementation:**
  - Uses a binary search to determine the threshold z^*.
  - Sets $m_k^* = \lfloor \mu_k / z^* \rfloor$ for each arm k.

2.3 `utils.py`

- **Purpose:** Provides helper routines, particularly for KL divergence, regret computation, and equilibrium verification.
- **Implementation:**
  - `kl_divergence(p, q)`: Computes the KL divergence between two Bernoulli distributions with parameters p and q. Used by SMAA for exploration checks.
  - `compute_instant_regret(arm_choices, env_mu, num_players)`: Computes the instantaneous regret for each player at the current round based on arm choices and expected rewards.
  - `is_equilibrium(arm_choices, env_mu, num_players)`: Checks whether the arm choices match the Nash equilibrium distribution computed from the environment's expected rewards.

2.4 `metrics.py`

- **Purpose:** Evaluates algorithm performance via regret and non-equilibrium metrics.
- **Implementation:**
  - `compute_regret(...)` calculates cumulative regret per player.
  - `compute_non_equilibrium_rounds(...)` counts how many rounds deviate from the paper's Nash Equilibrium allocation.

2.5 `player.py`

- **Purpose:** Implements the agent strategies, including the core SMAA algorithm.
- **Implementation:**

  - **SMAAPlayer** and **SMAAMusicalChairsPlayer**: Follow the block-based approach from the paper, periodically estimating the NE and using KL-based exploration.
  - Baselines (`ExploreThenCommitPlayer`, `TotalRewardPlayer`, `SelfishRobustMMABPlayer`) for comparison.

2.6 `simulation.py`

- **Purpose:** Runs experiments by orchestrating the environment and players over multiple rounds, tracking performance metrics, and generating visualizations.
- **Implementation:**
  - `run_simulation_time_series(...)`: Simulates a multi-player multi-armed bandit scenario over time, tracking arm choices, rewards, regret, and equilibrium status.
  - `paper_experiment(...)`: Runs the full-scale experiment as described in the paper, comparing multiple algorithms across different settings, seeds, and distributions.
  - `test_smaa_experiment(...)`: Conducts a small-scale experiment, demonstrating asymptotic performance bounds for different algorithms by tracking partial regrets and non-equilibrium rounds over time.
  - Allows partial (per-round) data collection to reproduce the paper's figures and final experimental results.

2.7 `plot_results.py`

- **Purpose:** Produces visualizations for cumulative regret, total rewards, and non-equilibrium rounds.
- **Implementation:**
  - `plot_regrets(...)` and `plot_non_equilibrium(...)` create bar charts for final metrics.
  - `plot_time_series_scenario(...)` draws time-series curves of regret and non-equilibrium, matching the paper's experimental style.

3 Proposed Extension: Time-Varying Rewards with Heterogeneous Player Preferences

3.1 Motivation

Real-world applications—such as online advertising, recommender systems, and financial trading—often feature dynamic reward distributions and heterogeneous player preferences. In these domains, the quality of an arm (resource) fluctuates over time due to factors like seasonality or market volatility, and different agents value the same resource differently. While the original SMAA framework assumes static rewards and uniform player valuations, extending the model to include time-varying rewards and personalized preferences makes it significantly more realistic and applicable.

3.2 Recall the assumptions of the Original Model

The original SMAA framework is built upon the following key assumptions:

- **Static Rewards:** Each arm k has a fixed expected reward μ_k drawn from a distribution (e.g., Beta or Bernoulli).
- **Homogeneous Players:** All players value each arm equally.
- **Averaging Allocation:** When multiple players select the same arm, they share the reward equally.
- **No-Ties Assumption:** For any two arms k_1 and k_2 and for any positive integers n_1, n_2, $\mu_{k_1}/n_1 \neq \mu_{k_2}/n_2$ (which holds with probability 1 when μ_k are drawn from a continuous distribution).

These assumptions facilitate a theoretical analysis of regret and equilibrium but do not capture fluctuations in resource quality or differences in players' valuations.

3.3 Our Extended Problem Setup

In our extended framework, we relax two key assumptions of the original model to better capture real-world complexity:

1. **Time-Varying Rewards:** Instead of static μ_k, we model the expected reward of each arm as a time-dependent function, $\mu_k(t)$. For instance, we may update the reward using a rule such as:

$$\mu_k(t+1) = \mu_k(t) + \Delta_k \cdot f(t),$$

   where Δ_k is a drift factor and $f(t)$ is a periodic function (e.g., a sine wave) that captures fluctuations over time. for example:

$$\mu_k(t+1) = \mu_k(t) + \Delta_k \cdot \sin\left(\frac{2\pi t}{50}\right).$$

2. **Heterogeneous Player Preferences:** We introduce a preference matrix $\Phi \in \mathbb{R}^{N \times K}$ where each entry $\Phi_{j,k}$ represents the valuation of arm k by player j. With this modification, the reward that player j receives when pulling arm k becomes:

$$R_{j,k}(t) = \frac{X_k(t)}{M_k(t)} \times \Phi_{j,k},$$

   where $X_k(t)$ is the observed reward and $M_k(t)$ is the number of players choosing arm k. This adjustment allows for personalized outcomes even when multiple players share the same arm.

3.4 Practical Significance

The extended model provides a more realistic representation of many practical systems. By incorporating dynamic rewards, the model can adapt to changes in resource quality over time—critical for environments with fluctuating user engagement or market conditions. The inclusion of heterogeneous preferences allows different players to receive personalized rewards, reflecting the fact that, in practice, the same resource may have varying values for different users. Together, these modifications enable the SMAA framework to more accurately mimic real-world scenarios, thereby improving its practical applicability in domains such as online advertising, recommendation systems, and financial markets.

3.5 Implementation Changes - Core Modifications

We modified several modules to support time-varying rewards and heterogeneous player preferences:

- **environment_dynamic.py:**
  - Implements periodic updates for $\mu_k(t)$ via `_update_arm_parameters(t)`, allowing rewards to fluctuate dynamically.
  - Maintains a **preference matrix** Φ to compute personalized rewards.
  - Defines **stochastic reward distributions** (e.g., Beta, Gaussian) around periodic reward functions $f_k(t)$.
- **player_dynamic.py:**
  - Uses **Exponential Moving Averages (EMA)** to update reward estimates dynamically.
  - Implements **adaptive equilibrium logic**, periodically recomputing the Nash equilibrium to reflect reward fluctuations.
  - Supports **self-regulating strategies** (e.g., `SelfishRobustMMABDynamic`), adjusting selection probabilities based on personal rewards.
- **metrics_dynamic.py:**
  - Tracks **time-dependent regret**, measuring deviation from optimal arm selection per time step.
  - Adjusts the **non-equilibrium metric** to track stability across rounds.
- **simulation_dynamic.py:**
  - Updates **arm parameters dynamically** within the main simulation loop.
  - Ensures players periodically update their **reward estimates and strategic choices** based on observed changes.

3.6 Algorithmic Adjustments

To improve adaptation in dynamic environments, we introduce:

- **Adaptive Exploration:**
  - Players periodically re-evaluate arms rather than committing long-term.
  - Implemented via **block-based exploration** or **change detection** in $\mu_k(t)$.
- **Moving Equilibrium Computation:**
  - Nash equilibrium is recomputed dynamically using the latest reward estimates.
  - Uses **binary search methods** to efficiently determine **stable allocations**.
- **Sliding-Window Updates:**
  - Maintains a **rolling window of recent observations** to adapt faster to periodic reward fluctuations.
  - Reduces noise impact and improves **short-term reward estimation**.

- **Strategic Player Adaptation:**
  - Players estimate **expected arm congestion** and adjust their selection probabilities accordingly.
  - Helps balance **exploration vs. exploitation** while **avoiding overcrowding**.

3.7 Open Challenges

Our implementation serves as a proof-of-concept, but further refinements are needed to improve theoretical soundness and empirical performance. Given the periodic nature of the policy, regret and equilibrium metrics must account for cyclical reward fluctuations, Heterogeneous Player Preferences and dynamic player adaptations.

4 Conclusion

Through this project, we explored the complex interactions among multiple agents within the multi-armed bandit (MAB) framework. Our work combined both theoretical analysis and practical implementation. In the theoretical aspect, we improved our skills in scientific literature review and interpretation through engagement with academic literature. In the practical part,we gained experience and improved our understanding of the connection between classes and the algorithm itself. Through the development process, we deepened our understanding of decentralized decision-making and learned how to connect theory with practice.This project enhanced our self-learning capabilities and motivated us to independently acquire new knowledge.We also gained additional experience in problem modeling. we considered integrating a specific problem into the model framework, but ultimately decided not to pursue this extension, and not to included in the final project.

Future Work In order to bridge the gap between theoretical model and practical applications, we suggest:

- Adversarial Robustness: Investigate how the SMAA algorithm can handle adversarial manipulation strategies.
- Dynamic Reward Environments: Adapt SMAA to settings where reward distributions vary over time, capturing real-world dynamics.
- Heterogeneous Player Preferences: Extend the model to incorporate diverse agent valuations, moving beyond the homogeneous assumption.
- Hybrid Approaches: Explore integrating hybrid methodologies, such as deep learning techniques, to enhance performance and robustness.

References

[1] Bande, M., Magesh, A., and Veeravalli, V. V. Dynamic spectrum access using stochastic multi-user bandits. *IEEE Wireless Communications Letters*, 10(5):953–956, 2021.

[2] Besson, L. and Kaufmann, E. Multi-player bandits revisited. In *Algorithmic Learning Theory*, pages 56–92. PMLR, 2018.

[3] Ben-Porat, O., Rosenberg, I., and Tennenholtz, M. Content provider dynamics and coordination in recommendation ecosystems. *Advances in Neural Information Processing Systems*, 33:18931–18941, 2020.

[4] Jagadeesan, M., Wei, A., Wang, Y., Jordan, M., and Steinhardt, J. Learning equilibria in matching markets from bandit feedback. *Advances in Neural Information Processing Systems*, 34:3323–3335, 2021.

[5] Jagadeesan, M., Garg, N., and Steinhardt, J. Supply-side equilibria in recommender systems. *arXiv preprint arXiv:2206.13489*, 2022.

[6] Liu, L. T., Mania, H., and Jordan, M. Competing bandits in matching markets. In *International Conference on Artificial Intelligence and Statistics*, pages 1618–1628. PMLR, 2020.

[7] Riahi, S. and Riahi, A. Game theory for resource sharing in large distributed systems. *International Journal of Electrical & Computer Engineering (2088–8708)*, 9(2), 2019.

[8] Rosenski, J., Shamir, O., and Szlak, L. Multi-player bandits – a musical chairs approach. In *International Conference on Machine Learning*, pages 155–163. PMLR, 2016.

[9] Roughgarden, T. *Selfish routing and the price of anarchy.* MIT Press, 2005.

[10] Shi, C. and Shen, C. Multi-player multi-armed bandits with collision-dependent reward distributions. *IEEE Transactions on Signal Processing*, 69:4385–4402, 2021.

Appendix: SMAA Pseudocode

pseudocode for the SMAA algorithm, which captures its main components .

Algorithm 1 SMAA Algorithm

```
1: Input: $K$ arms, $N$ players, horizon $T$.
2: Initialization:
     • Compute $K' \leftarrow N \cdot \lceil K/N \rceil$.
     • For $t = 1$ to $K'$: Each player pulls every arm once to obtain initial estimates.
3: for $t = K' + 1$ to $T$ do
4:     for each player $j$ do
5:         (a) Update Statistics:
               $$\tau_{j,k}(t) = \sum_{s=1}^{t-1} \mathbf{1}\{\pi_j(s) = k\}, \quad \hat{\mu}_{j,k}(t) = \frac{1}{\tau_{j,k}(t)} \sum_{s=1}^{t-1} \mathbf{1}\{\pi_j(s) = k\} X_k(s)$$
           for each arm $k$. (This step builds a reliable estimate of each arm's expected reward.)
6:         (b) Compute KL-UCB Indices:
               $$\hat{b}_{j,k}(t) = \sup\Big\{q \geq \hat{\mu}_{j,k}(t) : \tau_{j,k}(t)\,\mathrm{kl}\big(\hat{\mu}_{j,k}(t), q\big) \leq \log t + 4\log(\log t)\Big\}$$
           for each arm $k$. (These indices encourage exploration by quantifying uncertainty.)
7:         (c) Estimate Equilibrium Allocation:
               $$\hat{z}_j(t) = \sup\Big\{z > 0 : \sum_{k=1}^{K} \Big\lfloor \frac{\hat{\mu}_{j,k}(t)}{z} \Big\rfloor \geq N\Big\},$$
               $$\hat{m}_{j,k}(t) = \left\lfloor \frac{\hat{\mu}_{j,k}(t)}{\hat{z}_j(t)} \right\rfloor.$$
           (This approximates the ideal equilibrium allocation $m^*$.)
8:         (d) Build Selection List:
     •         For each arm $k$ with $\hat{m}_{j,k}(t) > 0$, compute the per-player reward:
               $$\tilde{r}_{j,k}(t) = \frac{\hat{\mu}_{j,k}(t)}{\hat{m}_{j,k}(t)}.$$
     •         Sort arms in descending order of $\tilde{r}_{j,k}(t)$ and construct the list $L_j(t)$ by repeating each arm $k$ exactly $\hat{m}_{j,k}(t)$ times.
           (The list $L_j(t)$ represents the player's current best estimate of the equilibrium allocation.)
9:         (e) Arm Selection:
     •         Compute index $i \leftarrow ((t + j) \mod N) + 1$.
     •         If $i = N$, then with probability 0.5, select a random arm outside the current estimated equilibrium set; otherwise, choose the $i$th element of $L_j(t)$.
     •         Otherwise (if $i \neq N$), choose the $i$th element of $L_j(t)$.
10:        Pull the chosen arm $\pi_j(t)$ and observe the reward $R_j(t)$ and arm reward $X_{\pi_j(t)}(t)$.
11:    end for
12: end for
```