Doubly Fair Dynamic Pricing

BS2308, BS2342, BS2344, BS2353

Introduction to Programming and Data Structures

December 21, 2023

Background

- The classic online pricing problem is described, where a seller proposes prices without knowing customers' valuations and receives binary feedback based on acceptance/rejection.
- The goal is to maximize expected revenue by learning from the feedback and improving knowledge of the demand distribution

Pricing problem

Online pricing. For $t = 1, 2, \ldots, T$:

1. The customer valuates the product as y_t.
2. The seller proposes a price v_t concurrently without knowing y_t.
3. The customer makes a decision $1_t = \mathbb{I}(v_t \leq y_t)$.
4. The seller receives a reward (revenue) $r_t = v_t \cdot 1_t$.

Dynamic pricing is a strategy where businesses adjust the prices of their products or services based on real-time market demand, supply, and other relevant factors allowing for flexible pricing that can change rapidly, optimizing revenue and responding to fluctuations in the market.

Introduction

Imagine you're running an online store with two types of customers:

- **Group A:** generally pay more for your products.
- **Group B:** tend to be budget-conscious and pay less.

Your goal is to set prices that:

1. Make you the most money (*Maximize Your Revenue*): This is called "maximizing revenue" or "optimizing profits."
2. Treat both groups fairly: Customers in Group A and Group B shouldn't have to pay significantly different prices for the same product just because they belong to different groups..

The Problem is:

- If you try to maximize revenue without thinking about fairness, you might end up charging Group A much more than Group B, even though they buy similar products. This is unfair.
- If you focus only on fairness and propose everyone the same, you might miss out on potential income from Group A customers who are willing to pay more.

Introduction

FPA comes in to solve this dilemma: Think of it as a smart assistant who helps you set prices. FPA divides time into "epochs" (like rounds in a game). In each epoch, it does 3 main things:

1. Gathers information: It experiments with different prices to see how customers in each group react.
2. Chooses "good" pricing options: It picks prices that both make you money and are fair to both groups.
3. Learns from its mistakes: It uses the information gathered in each epoch to improve its price selection in future epochs.

As a result: Over time, FPA gets better and better at finding prices that are both profitable and fair. It guarantees that the difference between your actual revenue and the best possible revenue ("The Regret") is very small. It also ensures that both groups experience similar, *fair prices on average* (with a small chance of error).

LET'S CONSIDER AN EXAMPLE :

- Imagine you're booking a ride-sharing service during rush hour. With dynamic pricing, if there's high demand and limited drivers available, the ride's cost may increase. Conversely, during quieter times, prices could be lower. This flexible pricing model adapts to changing circumstances, ensuring that when demand is high, the service remains available maximizing revenue for the ride-sharing company
- Here,In our project,the Algorithm aims to dynamic pricing policy with nearly zero unfairness for two groups while proposing and accepting prices also.
- Customers form two disjoint groups, where 30% customers are in Group 1 and the rest 70% are in Group 2. For each price in \$0.625, \$0.7, \$1, customers in two groups have different acceptance rates:

Acceptance Rate	**\$0.625**	**\$0.7**	**\$1**
G1 (30%)	$3/5$	$1/2$	$1/2$
G2 (70%)	$4/5$	$4/5$	$1/2$

Valuation Vector

- Group 1: $V_1 = \begin{bmatrix} 0.625 \\ 1 \end{bmatrix}$
- Group 2: $V_2 = \begin{bmatrix} 0.7 \\ 1 \end{bmatrix}$

Probability Vector

- Group 1: $\Pi_1 = \begin{bmatrix} \frac{20}{29} \\ \frac{9}{29} \end{bmatrix}$
- Group 2: $\Pi_2 = \begin{bmatrix} \frac{25}{29} \\ \frac{4}{29} \end{bmatrix}$

The only way to ensure fairness for both constraints is by proposing the same price for both groups. The optimal price is \$1, yielding an expected revenue of \$0.5, as depicted in the figure.
However, if we opt for a random price distribution for each group and examine fairness in expectation, we might find distributions satisfying both fairness constraints while achieving higher expected revenue than any fixed-price strategy.
consider the following random policy:

- For customers from G1, propose \$0.625 with a probability of 20/29 and \$1 with a probability of 9/29.
- For customers from G2, propose \$0.7 with a probability of 25/29 and \$1 with a probability of 4/29.

Under this policy, the expected proposed price and accepted price in both groups are \$43/58 and \$8/11, respectively. Furthermore, the expected revenue is \$74/145, surpassing \$0.5.
The graph below illustrates the expected revenue functions for prices in two groups. The red dashed line represents their weighted average by population.

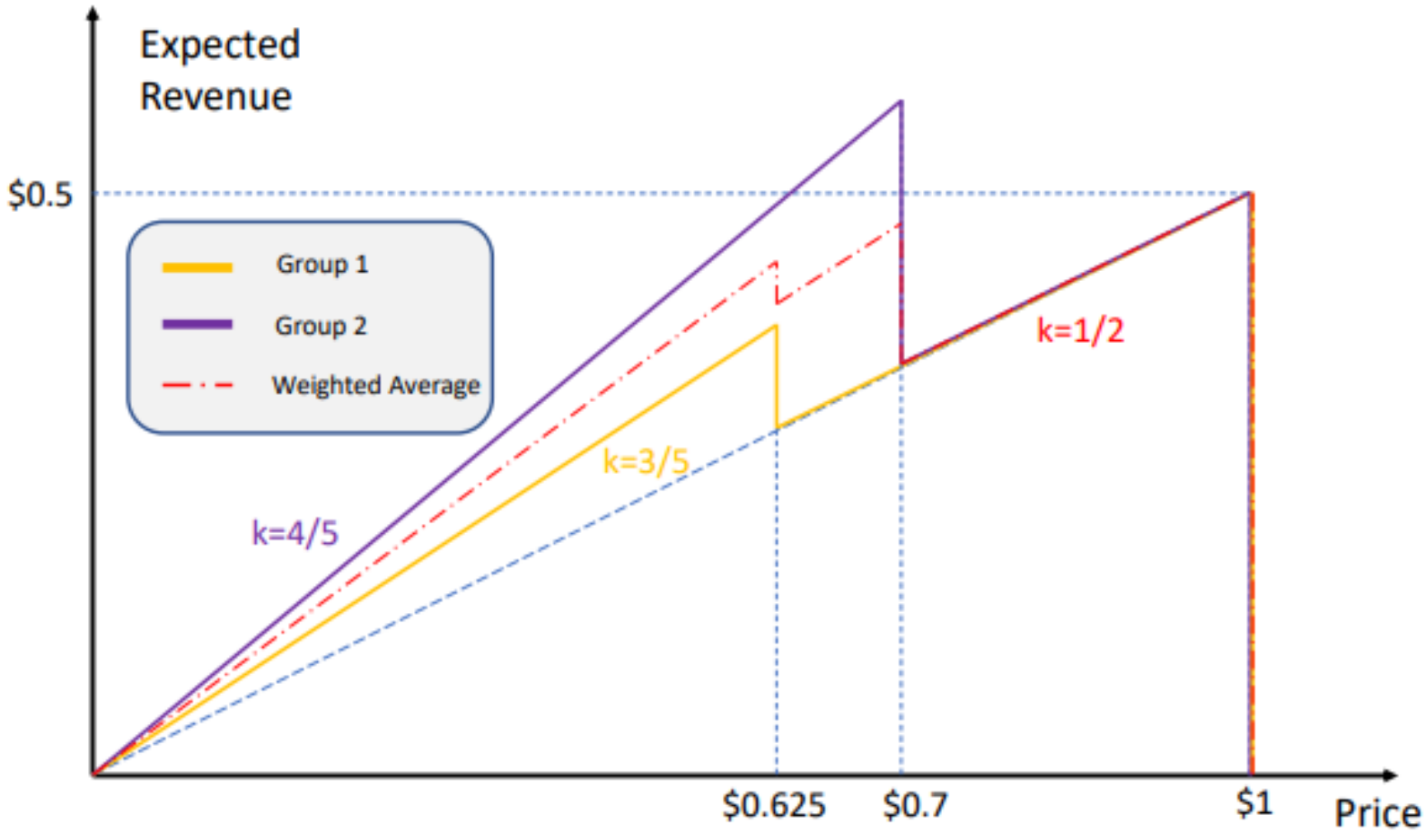

Figure 1: Expected Revenue vs Price

Introduction

- Problem Setting: Online Pricing and Fairness Definitions.
- Goal: Design an algorithm for optimal pricing under fairness constraints.

Problem Description

- Selling session with customers from groups G_1 and G_2 over T rounds.
- Prices chosen from set $\mathrm{V} = \{v_1, v_2, ..., v_d\}$.
- Pricing policy $\pi = \{\pi^1, \pi^2\}$ with distributions $\pi^1, \pi^2 \in \Delta^d$.
- Customer arrives with group attribution G_e, price proposed via π^e.
- Valuation y_t^e generated secretly, feedback 1_t^e, and reward r_t^e.

Definitions

Define:

- $\mathrm{v} = [v_1, v_2, ..., v_d]^\top$,
  $[d] = \{1, 2, ..., d\}$,
  $\mathbb{I} = [1, 1, ..., 1]^\top$.
- $F_e(i)$: Probability of price v_i being accepted in G_e.
  $F_e(i) = \Pr_{\mathbb{D}_e}[y_t^e \geq v_i]$
- $F_e = \mathrm{diag}\left(F_e(1), F_e(2), ..., F_e(d)\right)$.

Formulation

For a customer from G_e:

- Expected proposed price: $\mathbf{v}^\top \pi^e$.
- Expected reward: $\mathbf{v}^\top F_e \pi^e$.
- Expected acceptance rate: $\mathbb{I}^\top F_e \pi^e$.
- Expected accepted price: $\frac{\mathbf{v}^\top F_e \pi^e}{\mathbb{I}^\top F_e \pi^e}$.

Expected Revenue (Definition 2)

For any pricing policy $\pi = (\pi^1, \pi^2) \in \Pi$, expected revenue is the weighted average of rewards for G_1 and G_2:

$$R(\pi; F_1, F_2) = q \cdot \mathrm{v}^\top F_1 \pi^1 + (1 - q) \cdot \mathrm{v}^\top F_2 \pi^2.$$

Procedural Unfairness (Definition 3)

Procedural unfairness of a pricing policy $\pi \in \Pi$ is the absolute difference in expected proposed prices between groups:

$$U(\pi) = |\mathsf{v}^\top \pi^1 - \mathsf{v}^\top \pi^2| = |\mathsf{v}^\top (\pi^1 - \pi^2)|.$$

Substantive Unfairness (Definition 4)

Substantive unfairness of a pricing policy $\pi \in \Pi$ is the difference in expected accepted prices between groups:

$$S(\pi; F_1, F_2) = \left| \frac{\mathbf{v}^\top F_1 \pi^1}{\mathbb{I}^\top F_1 \pi^1} - \frac{\mathbf{v}^\top F_2 \pi^2}{\mathbb{I}^\top F_2 \pi^2} \right|.$$

Optimal Policy (Definition 5)

The optimal policy $\pi_* = \underset{\pi=(\pi^1,\pi^2)\in\Pi}{\text{argmax}} \; R(\pi; F_1, F_2)$ satisfies constraints $U(\pi) = 0$ and $S(\pi; F_1, F_2) = 0$.

Cumulative Regret (Definition 5)

The cumulative regret of algorithm $\mathcal{A}$ is defined as:

$$\mathrm{Reg}_T(\mathcal{A}) = \sum_{t=1}^{T} \left(R(\pi_*; F_1, F_2) - R(\pi_t; F_1, F_2) \right).$$

Cumulative Substantive Unfairness

The cumulative substantive unfairness of algorithm $\mathcal{A}$ is defined as:

$$S_T(\mathcal{A}) = \sum_{t=1}^{T} S(\pi_t; F_1, F_2).$$

Goal of Algorithm Design

The goal is to approach π_* in performance. The challenge is ensuring optimal regret while minimizing cumulative procedural unfairness and substantive unfairness over time.

Technical Assumptions

- **Assumption 1:** There exists a fixed constant $F_{\min} > 0$ such that $F_e(d) \geq F_{\min}$ for $e = 1, 2$. This ensures the soundness of the expected accepted price definition.
- **Assumption 2:** We treat d, the number of prices, as independent from T. Also, we assume $d = O(T^{1/3})$. This separation of dependence on d allows us to focus on the impact of fairness constraints on the pricing process over time and demonstrates the algorithm's optimality with respect to T.

Fairly Pricing Algorithm (FPA)

- **Purpose:** Identify a pricing policy maximizing revenue while ensuring procedural and substantive fairness.
- **Inputs:**
  - T: Time horizon :The duration of the pricing process.
  - V: Set of available prices :The set of prices that can be chosen.
  - ϵ: Error probability :The acceptable error in the algorithm.
  - L: Universal constant
  - q: Proportion :The proportion parameter likely used for fairness constraints.

Fairly Pricing Algorithm (FPA)

- **Before Epoch :** Propose highest price v_d to estimate $\hat{F}_{\min}$.
- Set $\Pi_1 = \Pi$: Policies with zero procedural unfairness.
- Set $I_0^1 = I_0^2 = [d]$: Include all prices.
- For each epoch $k = 1, 2, \ldots$:
  - Set parameters τ_k, $\delta_{k,r}$, $\delta_{k,s}$.
  - **Select good-and-exploratory policies** and form set A_k.
  - **Estimate acceptance probabilities** $F_e(i)$ using $\tilde{F}_{k,e}(i)$.
  - Construct matrices $\hat{F}_{k,e}$ based on estimates.
  - Find empirical optimal policy $\hat{\pi}_{k,*}$ using **Algorithm 2**.
  - Update policy set Π_k using **Algorithm 3**.

Algorithms

Before Epochs

In this stage, we keep proposing the highest price v_d for $\tau_0 = O(\log T)$ rounds to estimate (lower-bound) the least accepting probability $F_{\min}$.

Algorithm

- Initialize counters $M_{0,e} = 0$ and $N_{0,e} = 0$ for $e = 1, 2$.
- For $t = 1, 2, \ldots, \tau_0 = 2 \log T \log \frac{16}{\epsilon}$:
  - Denote the customer's group index as $e_t \in \{1, 2\}$.
  - Set $M_{0,e_t} += 1$.
  - Propose the highest price v_d.
  - If accepted, set $N_{0,e_t} += 1$.
- Calculate empirical acceptance rates:
  - $\frac{N_{0,1}}{2M_{0,1}}$ and $\frac{N_{0,2}}{2M_{0,2}}$.
- Output $\hat{F}_{\min} = \min\left\{\frac{N_{0,1}}{2M_{0,1}}, \frac{N_{0,2}}{2M_{0,2}}\right\}$.

Before Epoch Algorithm

```
from math import log
import random

T = 1000

v_max = 67
epsilon = 0.5
tau_0 = 2 * log(T) * log(16 / epsilon)

def estimate_min_acceptance_prob(v_max, tau_0, epsilon):
    N_accepted = {1: 0, 2: 0}
    N_proposed = {1: 0, 2: 0}

    for _ in range(int(tau_0)):
        customer_group = random.randint(1, 2)

        N_proposed[customer_group] += 1
        if accept_price(v_max):
            N_accepted[customer_group] += 1

    return min(N_accepted[1]/(2*N_proposed[1]),N_accepted[2]/(2*N_proposed[2]))

def accept_price(price):
    return random.random() < price

min_acceptance_prob = estimate_min_acceptance_prob(v_max, tau_0, epsilon)
print(f"Estimated minimum acceptance probability: {min_acceptance_prob}")
```

Doubling Epochs in FPA Algorithm

Purpose:

- Balance exploration and exploitation over time.
- Improve estimates of acceptance probabilities and refine pricing policies.

Key Points:

- Divide the time horizon (T) into epochs of increasing length.
- Epoch k has length $\tau_k = O(\sqrt{T} \cdot 2^k)$, doubling the previous epoch's length.
- Enables more exploration initially and more exploitation later.
- Longer epochs lead to better estimation of acceptance probabilities and policy improvement.
- Expect reduced regret in subsequent epochs due to better policy choices.

Good-and-Exploratory Policies

Purpose:

- Ensure sufficient exploration of different prices.
- Identify potentially profitable policies that encourage exploration.

Algorithm Steps:

1. **Initialization:**
   - Start with an empty set of good-and-exploratory policies ($A_k = \emptyset$).
   - Set I_k^e (index sets of prices to explore for group e) to the same as in the previous epoch.
2. **Finding Exploratory Policies:**
   - For each group ($e = 1, 2$) and price index (i) in I_k^e:
     - Find the policy ($\pi_{k,i,e}$) from the remaining policies (Π_k) that maximizes the probability of proposing price v_i in group e.
     - If this probability is at least $\frac{1}{\sqrt{T}}$, add the policy to A_k.
     - Otherwise, remove price v_i from I_k^e (no need to explore it further).
3. **Output:**
   - Return the set A_k of good-and-exploratory policies.

select good and exploratory policies

```python
import numpy as np

class Policy:
    def __init__(self, index, group):
        self.index = index
        self.group = group

    def calculate_probability(self, price):
        return np.random.rand()

def select_good_and_exploratory_policies(Pi_k, I_k_minus_1_1, I_k_minus_1_2, T):
    A_k = set()
    I_k_1 = I_k_minus_1_1.copy()
    I_k_2 = I_k_minus_1_2.copy()

    for group in [1, 2]:
        for price_index in range(len(I_k_minus_1_1)):
            i=I_k_minus_1_1[price_index] if group == 1 else I_k_minus_1_2[price_index]

            tilde_pi_k_i_e = max(Pi_k, key=lambda pi: pi.calculate_probability(i))

            if tilde_pi_k_i_e.calculate_probability(i) >= 1 / np.sqrt(T):
                A_k.add(tilde_pi_k_i_e)
            else:
                if group == 1:
                    I_k_1.remove(i)
                else:
                    I_k_2.remove(i)

    return A_k
```

Probability Estimates

Purpose:

- This algorithm estimates the acceptance probabilities of different prices for two groups of users (G1 and G2) within an epoch of a pricing algorithm.
- It's used to update the algorithm's understanding of user behavior and guide the selection of better pricing policies in subsequent epochs.

Algorithm Steps:

- Set counters $M_{k,e}(i)$ and $N_{k,e}(i)$ to 0 for all prices and groups.
- For each policy π in the set A_k of good-and-exploratory policies:
  - Run π for a batch of $\frac{\tau_k}{|A_k|}$ rounds.
  - Update counters based on observed prices and acceptances:
    - Increment $M_{k,e}(i)$ for each proposed price v_i in group G_e.
    - Increment $N_{k,e}(i)$ for each accepted price v_i in G_e.
- For each group e:
  - Calculate empirical acceptance probabilities $\bar{F}_{k,e}(i) = \max\left\{\frac{N_{k,e}(i)}{M_{k,e}(i)}, \hat{F}_{\min}\right\}$ for prices in I_k^e.
  - If $\bar{F}_{k,e}(i)$ is too low, use $\hat{F}_{\min}$ instead.
- Find a policy $\pi_k, *$ that maximizes expected revenue $R(\pi, \hat{F}_{k,1}, \hat{F}_{k,2})$ while ensuring $S(\pi, \hat{F}_{k,1}, \hat{F}_{k,2}) \leq \delta_{k,s}$.

Estimated Acceptance Probabilities

```
import numpy as np

class Policy:
    def __init__(self, index, group):
        self.index = index
        self.group = group

    def run_policy(self, rounds):
        # Actual logic to run the policy
        return [np.random.rand() for _ in range(rounds)]

def estimate_acceptance_probabilities(A_k, tau_k, I_k_1, I_k_2, F_hat_min):
A_k = {Policy(index=i, group=1) for i in range(5)}
tau_k = 1000
I_k_1 = [0, 1, 2, 3, 4]
I_k_2 = [0, 1, 2, 3, 4]
F_hat_min = 0.1
result = estimate_acceptance_probabilities(A_k, tau_k, I_k_1, I_k_2, F_hat_min)
```

Policy Eliminations: Removing Suboptimal Policies

- Remove policies that meet either of these criteria:
  - Large unfairness: Policies causing a significant difference in accepted prices between groups are considered undesirable. $S(\pi, \hat{F}_{k,1}, \hat{F}_{k,2}) > \delta_{k,s}$
  - Large regret: If the regret, denoted by $R(\pi, \hat{F}k, 1, \hat{F}k, 2)$, falls below a threshold determined by the optimal policy $\pi_{k,*}$. This considers the difference between the expected revenue of a policy and the optimal policy, accounting for estimation error ($\delta_{k,r}$) and potential increases in optimal reward due to relaxed fairness constraints ($L \cdot \delta_{k,s}$).
    $R(\pi, \hat{F}_{k,1}, \hat{F}_{k,2}) < R(\pi_k, *, \hat{F}_{k,1}, \hat{F}_{k,2}) - \delta_{k,r} - L \cdot \delta_{k,s}$

Algorithms 2

Purpose:

- Identify a policy maximizing expected revenue while respecting fairness constraints.
- Search for a policy balancing performance and fairness effectively.

Algorithm 2 Inputs

Inputs:

- $\hat{F}_{k,1}$ and $\hat{F}_{k,2}$ (estimated acceptance probability matrices)
- Π_k (candidate policy set)
- $\hat{F}_{\min}$ (lower bound on least accepting probability)
- $\delta_{k,s}$ (allowed estimation error for substantive unfairness)
- ϵ (step length for searching fair policies)

Algorithm 2 Steps

Steps:

1. **Initialization:**
   - Choose an arbitrary policy $\pi_{k,*}$ from Π_k as a starting point.
2. **Iteratively Solving Linear Programs:**
   - For each w_ℓ (increasing in steps of ϵ):
     - Break if w_ℓ exceeds a threshold based on $\hat{F}_{\text{min}}$.
     - Solve a linear program to find $\pi_{k,\ell,*}$ maximizing $R(\pi, \hat{F}_{k,1}, \hat{F}_{k,2})$ subject to procedural and substantive fairness constraints.
     - Update $\pi_{k,*}$ if $R(\pi_{k,\ell,*}, \hat{F}_{k,1}, \hat{F}_{k,2}) > R(\pi_{k,*}, \hat{F}_{k,1}, \hat{F}_{k,2})$.
3. **Return:**
   - Return the final empirical optimal policy $\pi_{k,*}$.

Algorithms 3

Purpose:

- Reduces the set of candidate policies: It eliminates policies that are unlikely to be optimal in terms of revenue and fairness, making the search for the best policy more efficient.
- Accelerates convergence: By removing suboptimal policies, it helps the algorithm find a fair and revenue-maximizing policy faster.

Algorithms 3

Input:

- $\hat{F}_{k,1}$, $\hat{F}_{k,2}$ (estimated acceptance probability matrices)
- Π_k (candidate policy set from the previous epoch)
- $\hat{F}_{\min}$ (lower bound on least accepting probability)
- $\delta_{k,s}$ (allowed estimation error for substantive unfairness)
- $\delta_{k,r}$ (allowed estimation error for revenue)
- L (constant related to performance-fairness tradeoff)
- $\hat{\pi}_{k,*}$ (empirical optimal policy found in Algorithm 2)
- ϵ (step length for searching for fair policies)

Algorithm 3 Steps

Initialization:

- Feasible set $\Pi_{k+1} = \emptyset$ for the next epoch.

Iteratively Finding Feasible Sets:

- For each $\ell = 0, 1, 2, \ldots$:
  - Let $w_\ell = \ell \cdot \epsilon$.
  - Break if $w_\ell > \frac{1}{\hat{F}_{\mathbf{min}}}$.
  - Solve a linear program to find a set of policies $\Pi_{k+1,w}$ satisfying various fairness and revenue constraints.
  - Update $\Pi_{k+1} \leftarrow \Pi_{k+1} \cup \Pi_{k+1,w}$.
  - Return the updated candidate policy set Π_{k+1} for the next epoch.

Conclusion

- **Problem Overview:**
  - Explored online dynamic pricing with dual fairness constraints.
  - Procedural fairness: Ensures equality in proposed prices across groups.
  - Substantive fairness: Ensures equality in accepted prices across groups.
- **Algorithm FPA:**
  - Utilizes random pricing policies for simultaneous procedural and substantive fairness.
  - Achieves $O(\sqrt{T})$ regret with optimal performance up to log(log(T) factors.
  - Guarantees zero procedural unfairness and maintains $O(\sqrt{T})$ substantive unfairness.
- **Significance:**
  - First algorithm to dynamically learn pricing while satisfying both fairness constraints simultaneously.
  - Balances fairness considerations with revenue optimization in online pricing.