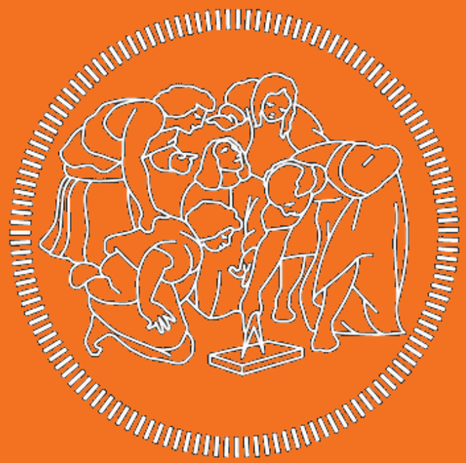
Online Learning Applications

“Pricing and Advertising strategy for e-Commerce of Flight Booking”

A.A. 2022-2023

Filippo Lazzarin, Alberto Pirillo, Michele Simeone, Simone Tognocchi



POLITECNICO
MILANO 1863

Team

Michele Simeone

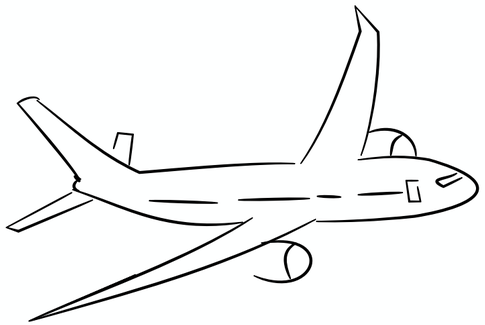
Simone Tognocchi

Alberto Pirillo

Filippo Lazzarin

Overview

- Problem setting
- Environment design
- Clairvoyant algorithm



Steps

1. Learning for pricing
2. Learning for advertising
3. Learning for joint pricing and advertising
4. Contexts and their generation
5. Dealing with non-stationary environments with two abrupt changes
6. Dealing with non-stationary environments with many abrupt changes

Problem setting

An **airline company** which sells plane tickets in its **e-commerce** platform wants to maximize its profit by dynamically learning an optimal pricing and advertising strategy.

Pricing: the same ticket could be sold at 5 different prices. The company wants to know the best one, i.e. the one that maximizes the **product of the conversion rate by the price**. The conversion rate represents whether a user who has seen the ticket will buy it.

Standard **Multi-Armed Bandits (MABs)** algorithms can be used to learn the conversion rates in an online fashion efficiently.

Advertising: the company wants to develop an **online advertising campaign** so that many users will discover it and see the services it offers by visiting the webpage of a **publisher.** The company has to select a **bid** (the maximum amount it is willing to pay the publisher). Selecting a higher bid will result in more users seeing the advertisement but also more costs charged by the publisher to the company.

Gaussian Process (GP) Bandits algorithms can be used to dynamically learn the optimal bid while minimizing the loss in the meantime.

Environment Design Classes

Based on the information that we retrieve from the purchase of the ticket we can divide customers into different segments.

The features that we have chosen to classify the users are:

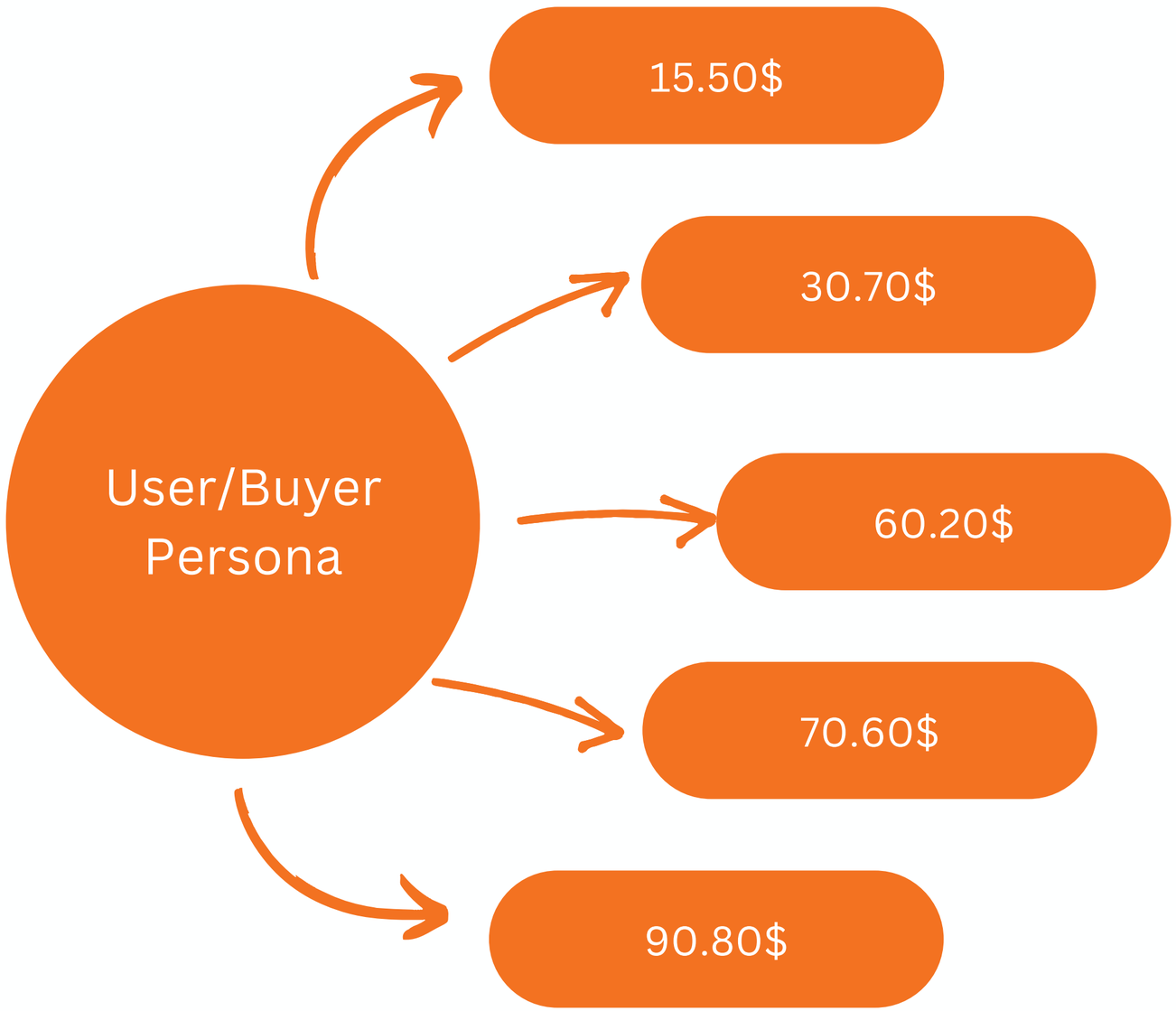
- Whether a customer chooses to buy **checked luggage**
- Whether a customer chooses to buy a **first-class ticket**

Based on these features we created three different contexts:

- First class of users is represented by those **under 25** years
- Second class by adult users with an age in the **range of 25-40** years
- Third class by those **over 40** years

The **under 25** do not pick up any feature for their ticket.
The **adult (25-40)** users pick up just one between checked luggage and a first-class ticket.
The **over 40** choose both features when purchasing the ticket.

Classes and Prices



Environment Design
Conversion Rates

The flight market is affected by **seasonality** which subsequently influences the conversion rates.

In general, we can split the time horizon (**365 days**) into **three periods**:

- The **winter period** goes from October to January
- The **spring period** from February to May
- The **summer period** from June to September

Usually, tickets tend to have the **highest conversion rates during summer**, while conversion rates tend to be slightly lower during winter and are the **lowest in the spring**.

When we decided on the conversion rates we tried to be as faithful as possible to **real-world** case scenarios while using values that allow for easily visualizable results.

Clairvoyant algorithm

The **objective function** to maximize is defined as the **reward**.

- For one class the **reward** is defined as:

(number of daily clicks * conversion rate * margin) - cumulative daily cost

- The **margin** is the difference between the price at which the product is sold and its corresponding cost. For simplicity, we consider a cost of 0 for each product.
- For **multiple classes**, the reward will be the **sum** of the rewards provided by the single classes.

We have approximated the continuous set of bids with a finite set of 100 bids.

The optimization algorithm works as follows:

- Find the best price for every single class, independently from the others
- Optimize the bid for each class independently from the other classes

Therefore, the algorithm requires **two exhaustive searches:**

- Over the prices for every class
- Over the bids for every class

Thus, the algorithm runs in linear time in the number of prices, bids, and classes.

Experiment parallelization

When working with **stochastic processes**, it is required to run multiple simulations of the same scenario in order to **smooth out the stochasticity**. We refer to these simulations as *experiments*.

The **proper number of experiments** depends on the environments and on the algorithms used. We found that pricing algorithms (e.g. UCB1, TS) required much more experiments w.r.t. advertising algorithms (e.g. GPU-UCB, GP-TS).

Every experiment is **independent** from the others. Therefore, we can **run multiple experiments in parallel** on different CPU cores, using the *map* function of the *multiprocessing* Python module.

Essentially, we **encapsulated the entirety of an experiment inside of a single function**. Then, the function is executed in parallel on different cores. In the end, the results are collected in a common data structure.

Observed speedups are in the range of **5x to 7x** depending on the number of available CPU cores.
Parallelization was the reason why we were able to perform this many experiments.

Step 1: Learning for pricing

In this scenario:

- All users belong to a **single class** (let's call it *C1*)
- Information about the **advertising** part of the problem is **known**
- Information about the **pricing** part is **lacking**

To tackle this scenario and learn online information about the pricing part, we will apply two algorithms: **UCB1** and **Thompson Sampling**.

UCB1

Key Idea:
UCB1 balances the exploration-exploitation trade-off, trying to both explore the arms with uncertain rewards and exploit arms that seem promising.

Upper Confidence Bound (UCB):

- For each arm, UCB1 calculates an upper confidence bound representing the uncertainty in the estimated reward
- This upper bound is based on the arm's historical performance and a confidence level parameter

Benefits:

- UCB1 is simple, efficient, and widely used in various applications, such as online advertising, clinical trials, and recommendation systems
- It strikes a balance between exploration and exploitation, leading to good overall performance

Thompson Sampling (TS)

Key Idea:
Thompson Sampling takes a Bayesian approach to the problem, modeling the uncertainty about the reward probabilities for each arm.

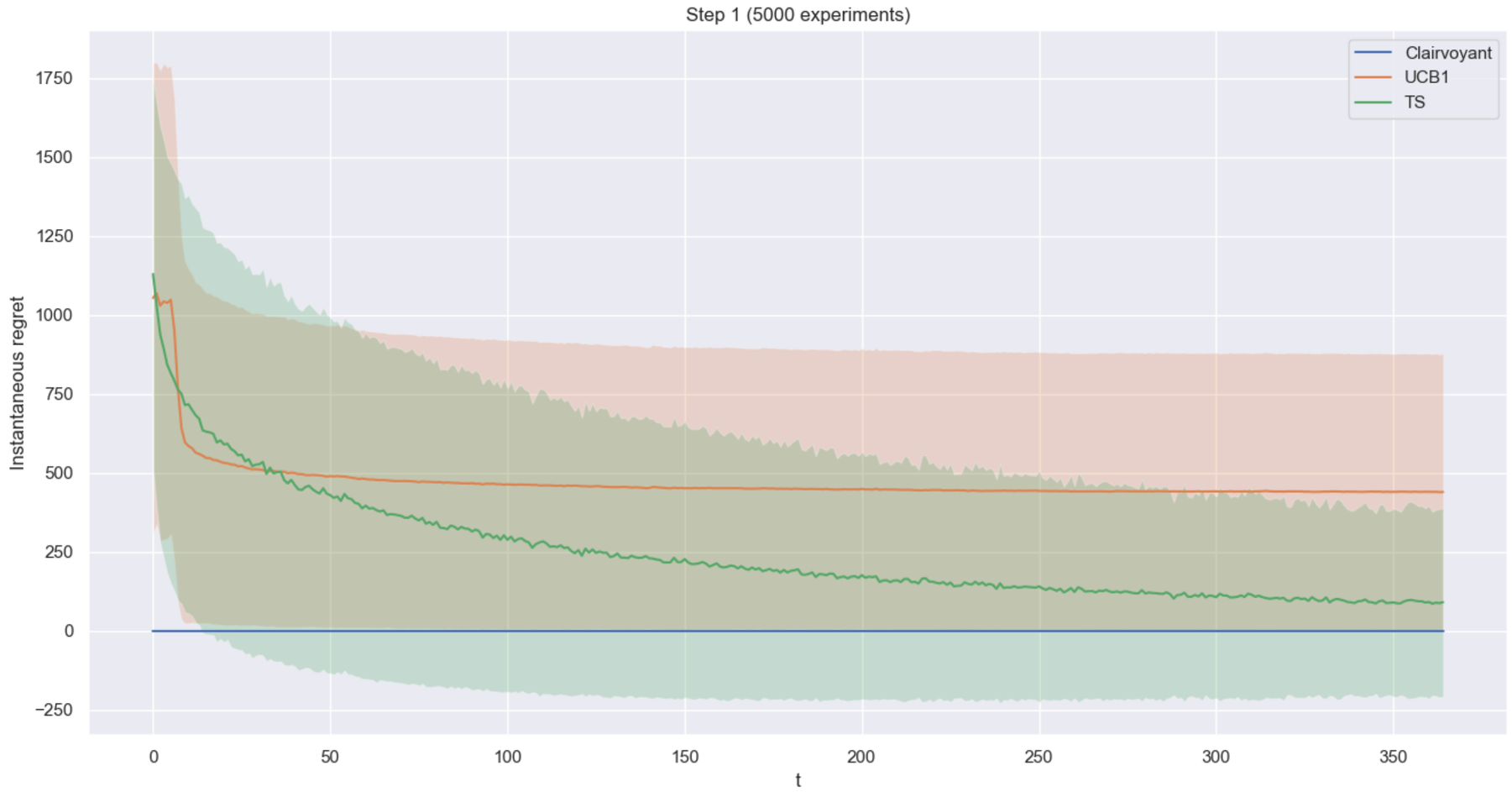
Probability Distributions:

- For each arm, Thompson Sampling maintains a probability distribution (usually a **Beta** distribution) over possible reward probabilities.
- These distributions represent our beliefs about the arms' performance.

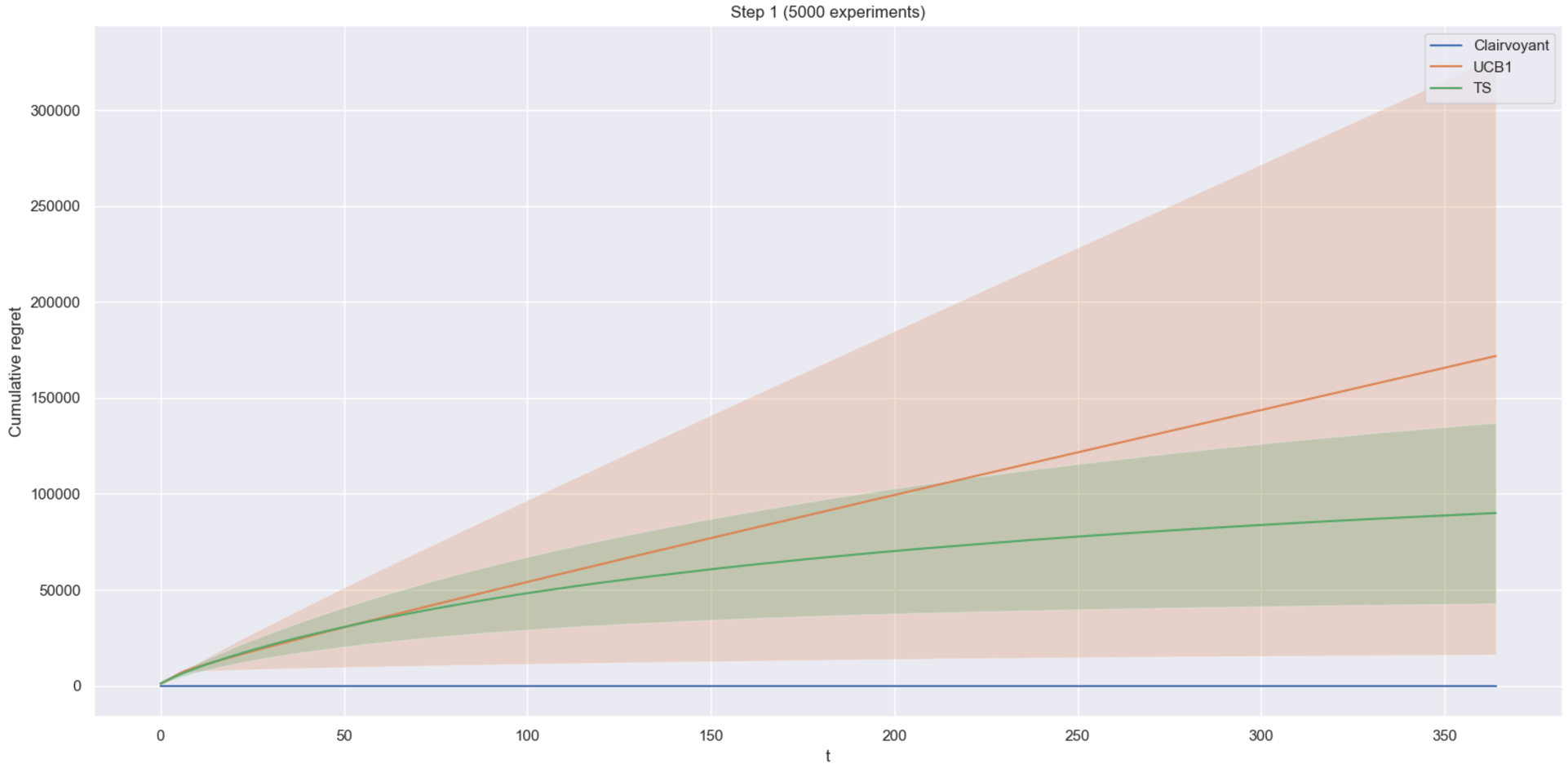
Benefits:

- Thompson Sampling is effective in situations where the reward probabilities can change over time or exhibit complex patterns
- It adapts quickly to emerging trends and uncertainties in the environment

Instantanenous Regret



Cumulative Regret



Step 2: Learning for advertising

This scenario is the same as Step 1, but this time:

- The **pricing** part is **known**
- The **advertising** part is **not** known

To tackle this scenario, we will apply two algorithms: **GP-UCB** and **GP-TS**, both of which rely on Gaussian Processes (GPs) to model the uncertainty in the advertising curves.

Gaussian Process

GP-UCB and GP-TS utilize Gaussian Processes, a **probabilistic modelling technique**, to represent the uncertainty about the arms' reward distributions.

GPs provide a flexible framework for **modelling complex, unknown functions**, with very mild assumptions.

GP-UCB & GP-TS

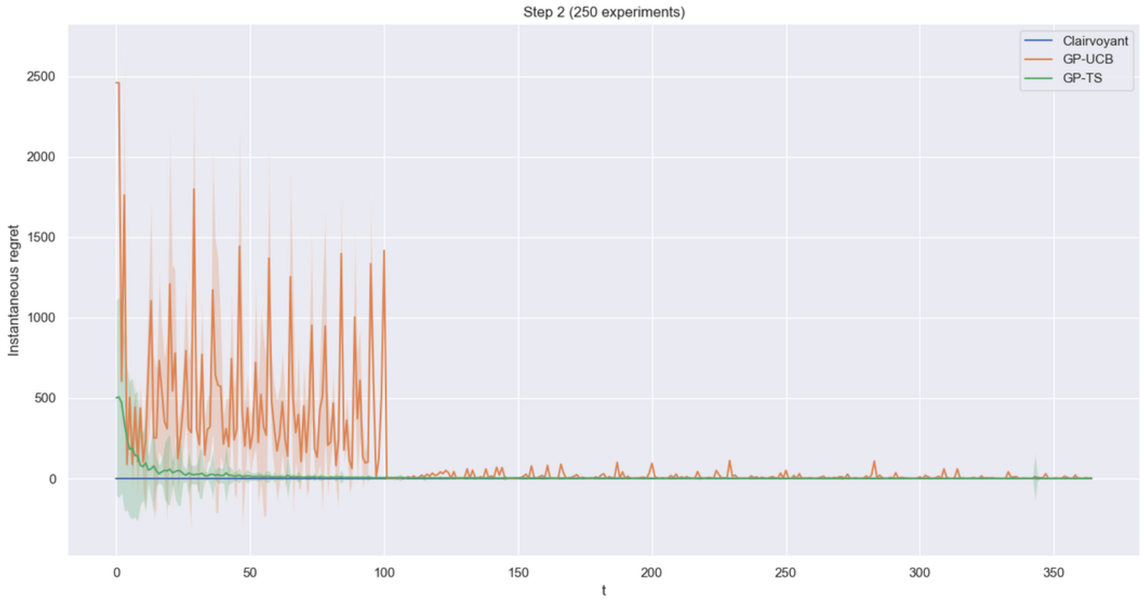
Considerations:

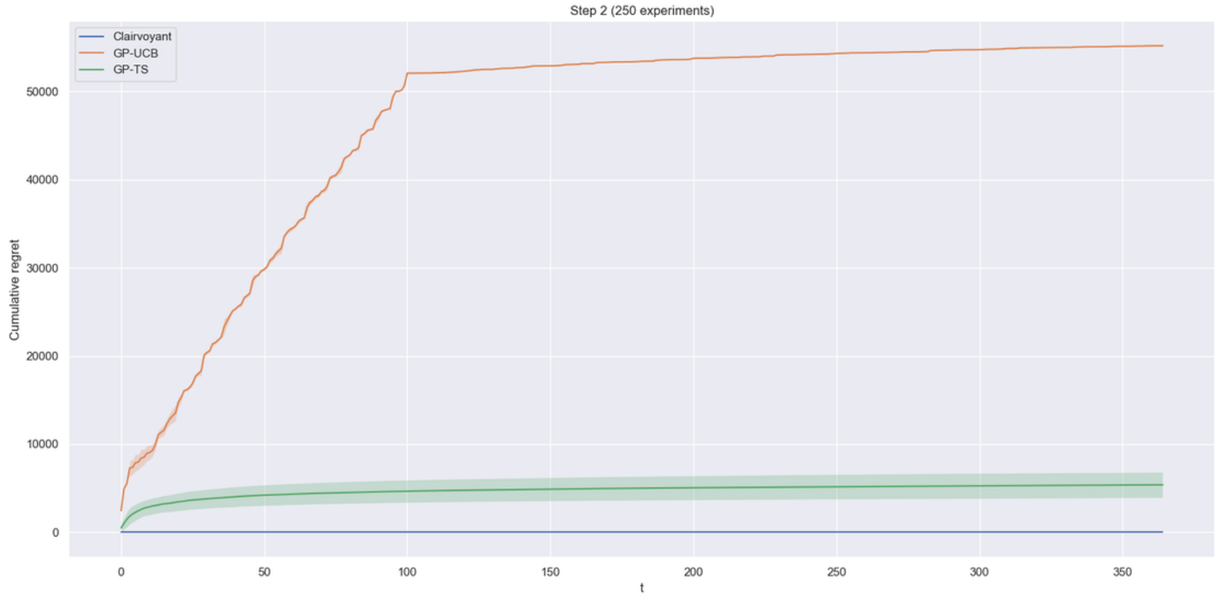
- The choice of GP hyperparameters and the kernel function significantly impact algorithm performance
- GP-MAB assumes that the reward distribution remains stationary, which may not hold in dynamic environments

Performance:

GP-MAB has been widely applied in various domains and has demonstrated strong empirical performance

Instantaneous Regret & Cumulative Regret



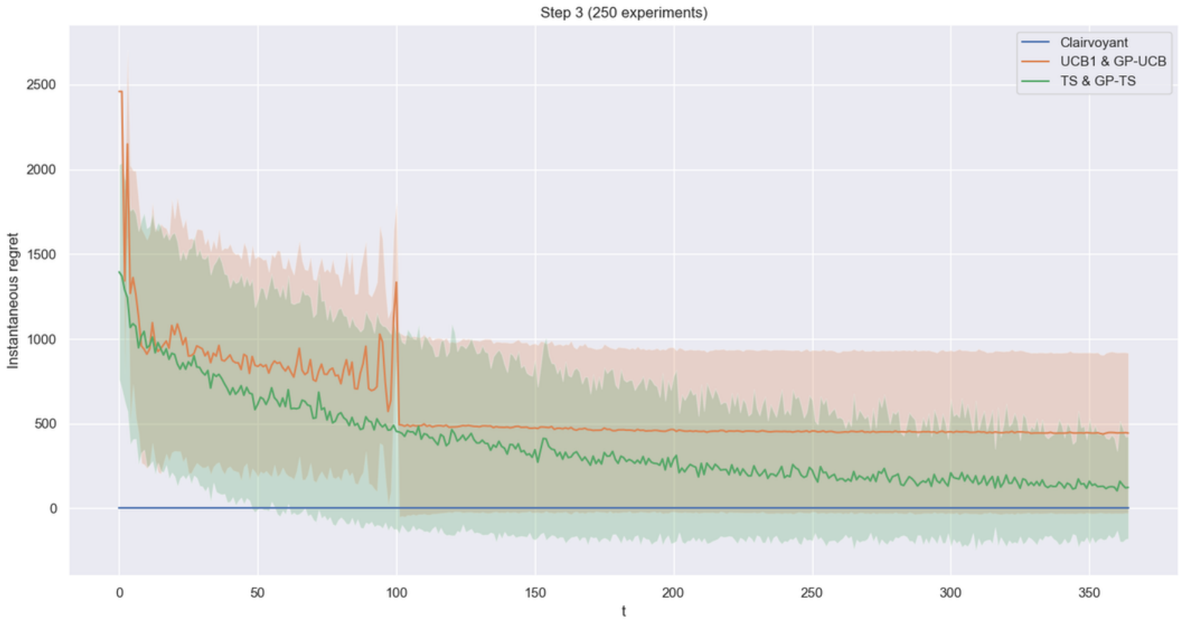


Step 3: Learning for joint pricing and advertising

This scenario is the same as Step 1, but this time we have **no prior knowledge** on either the pricing part or the advertising part.

To tackle this scenario, we will **combine** the algorithms of the previous 2 steps. As a result, we will need to run the two algorithms **sequentially**, using the estimate computed by the pricing algorithms to compute the estimate for the advertising algorithms.

Instantaneous Regret & Cumulative Regret



Step 4: Contexts and their generation

In a complex situation, we have three different groups of users **(C1, C2, and C3)**, and we **lack** any prior information about how advertising and pricing strategies should be applied to them. We are exploring two scenarios:

1. **Scenario 1:** we already know the structure of the user contexts so we understand how different users are grouped or categorized beforehand.
2. **Scenario 2:** In the second scenario, we do **not have prior** knowledge of the context structure. We need to **learn** this structure from the available data. It's important to note that in this scenario, **we don't even know how many user contexts exist.**

To address both scenarios, we will use two algorithms: GP-UCB and GP-TS. Additionally in Scenario 2, we will pair these algorithms with a context generation algorithm.

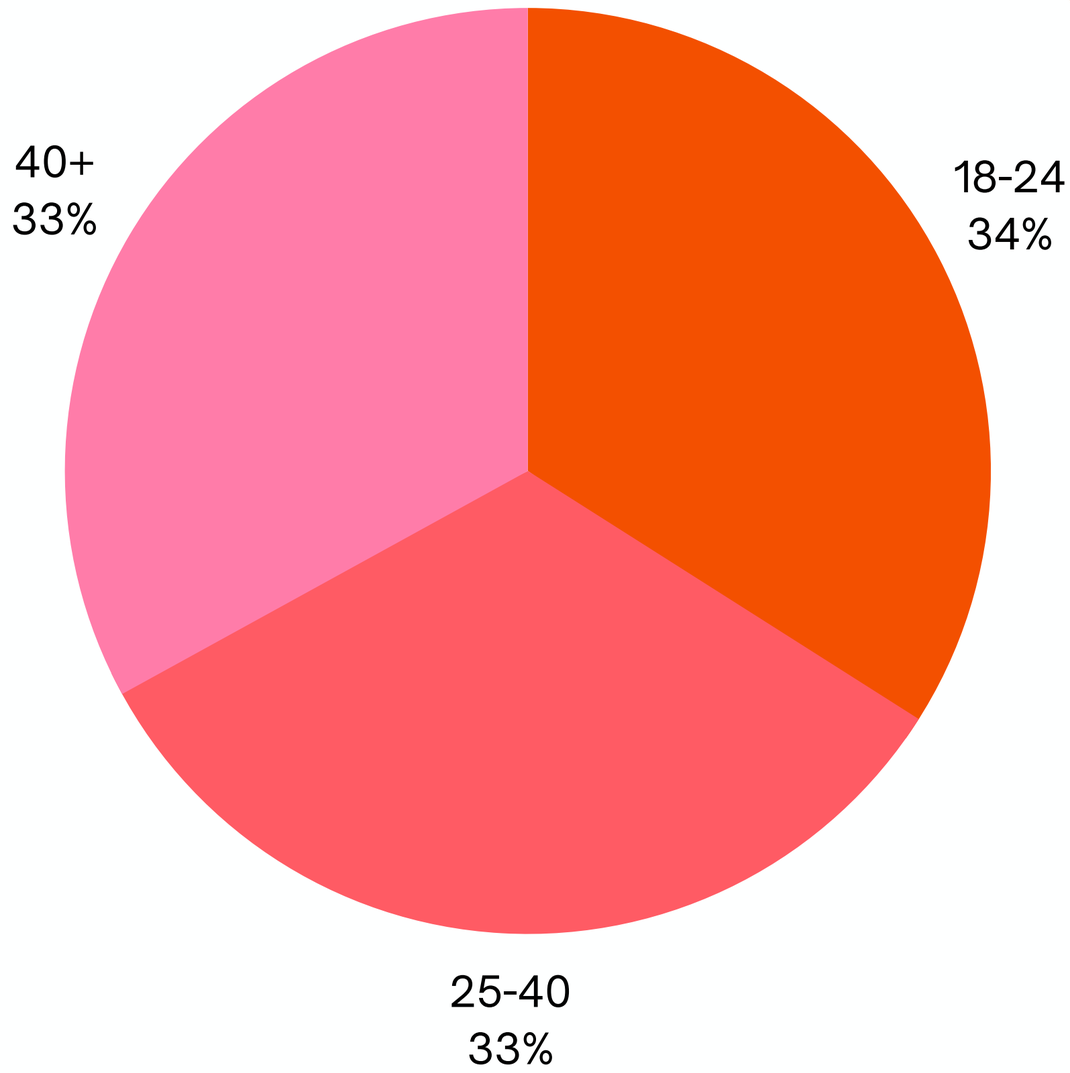
At the end, we will also run GP-UCB and GP-TS algorithms without context generation. We'll treat all users as if they belong to a single context for the entire duration of the analysis.

Classes

01 18-24

02 25-40

03 40+



Split Condition& Hoeffding Bound

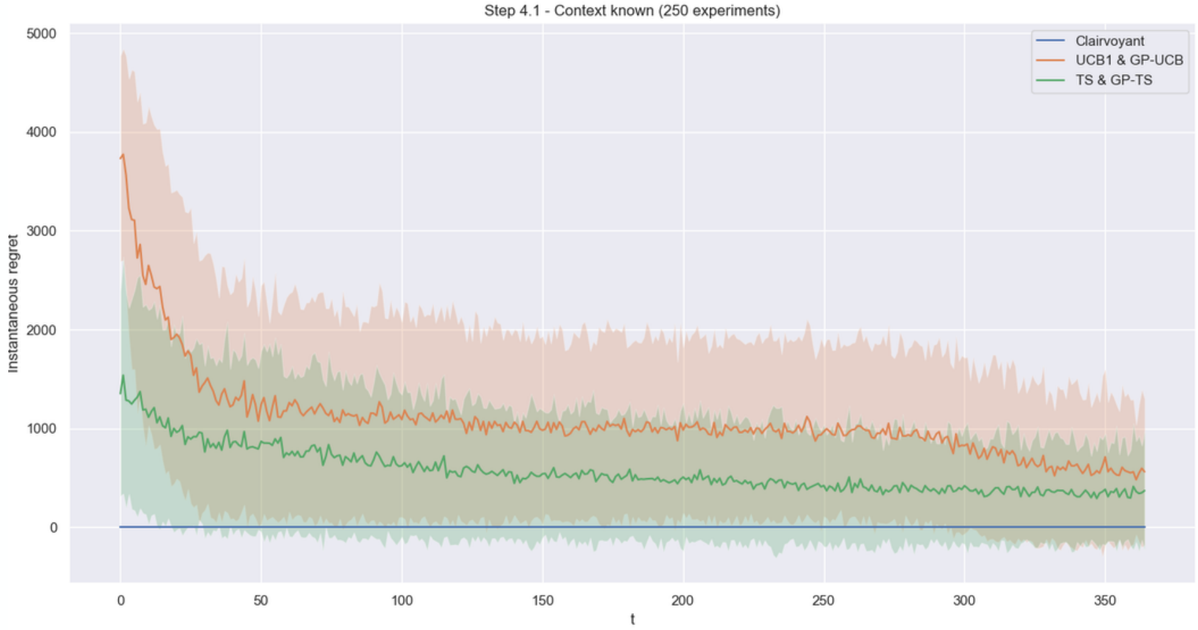
After calculating the value of the **best arms** in each testing context, we select the maximum between the **lower bound weighted sum** of the classes belonging at each context.

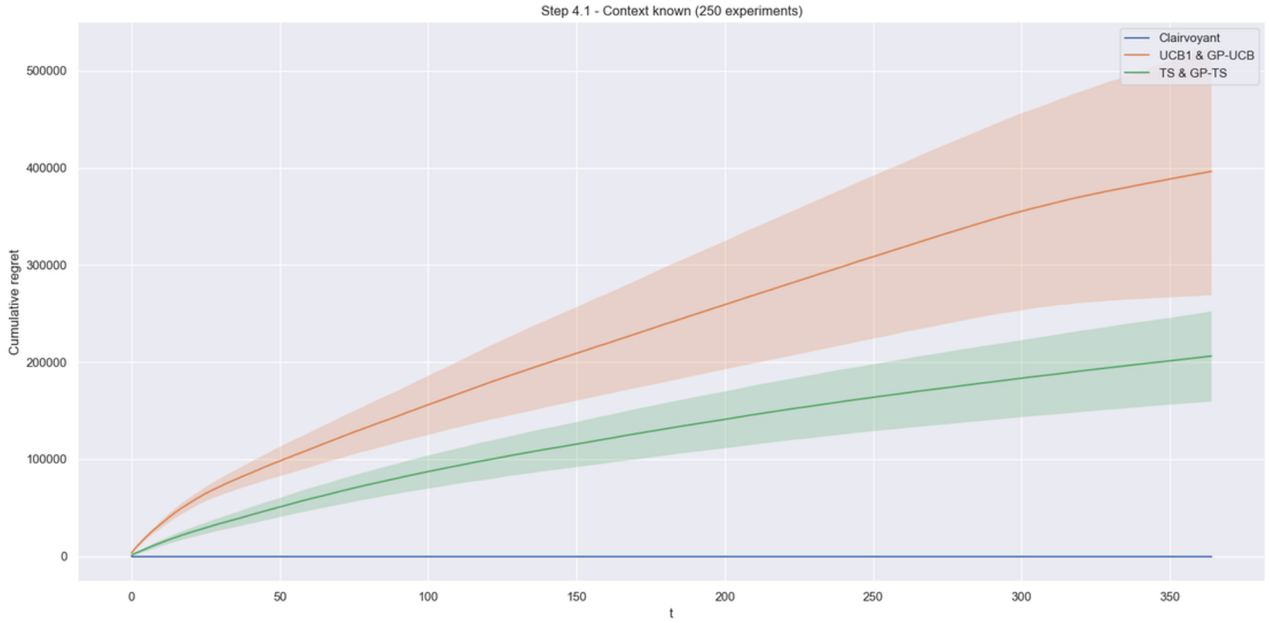
$$\underline{p}_{c_1}\,\underline{\mu}_{a^*_{c_1},c_1} + \underline{p}_{c_2}\,\underline{\mu}_{a^*_{c_2},c_2} \geq \underline{\mu}_{a^*_{c_0},c_0}$$

The **Hoeffding Bound** is a useful tool for estimating lower bounds on probabilities when working with limited sample data. It provides a statistically sound method to calculate lower bounds with a specified level of confidence, making it valuable in various data-driven applications.

$$\bar{x} - \sqrt{-\frac{\log(\delta)}{2|Z|}}$$

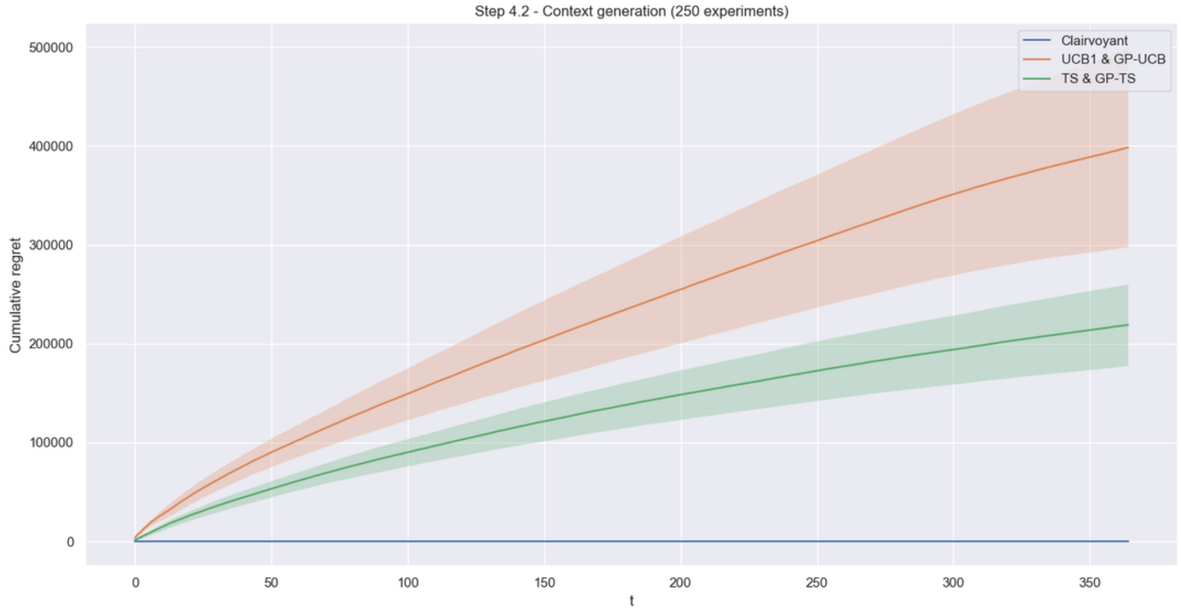
4.1 - Regret


Step 4.1 - Context known (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Instantaneous regret
t
5000
4000
3000
2000
1000
0
0
50
100
150
200
250
300
350

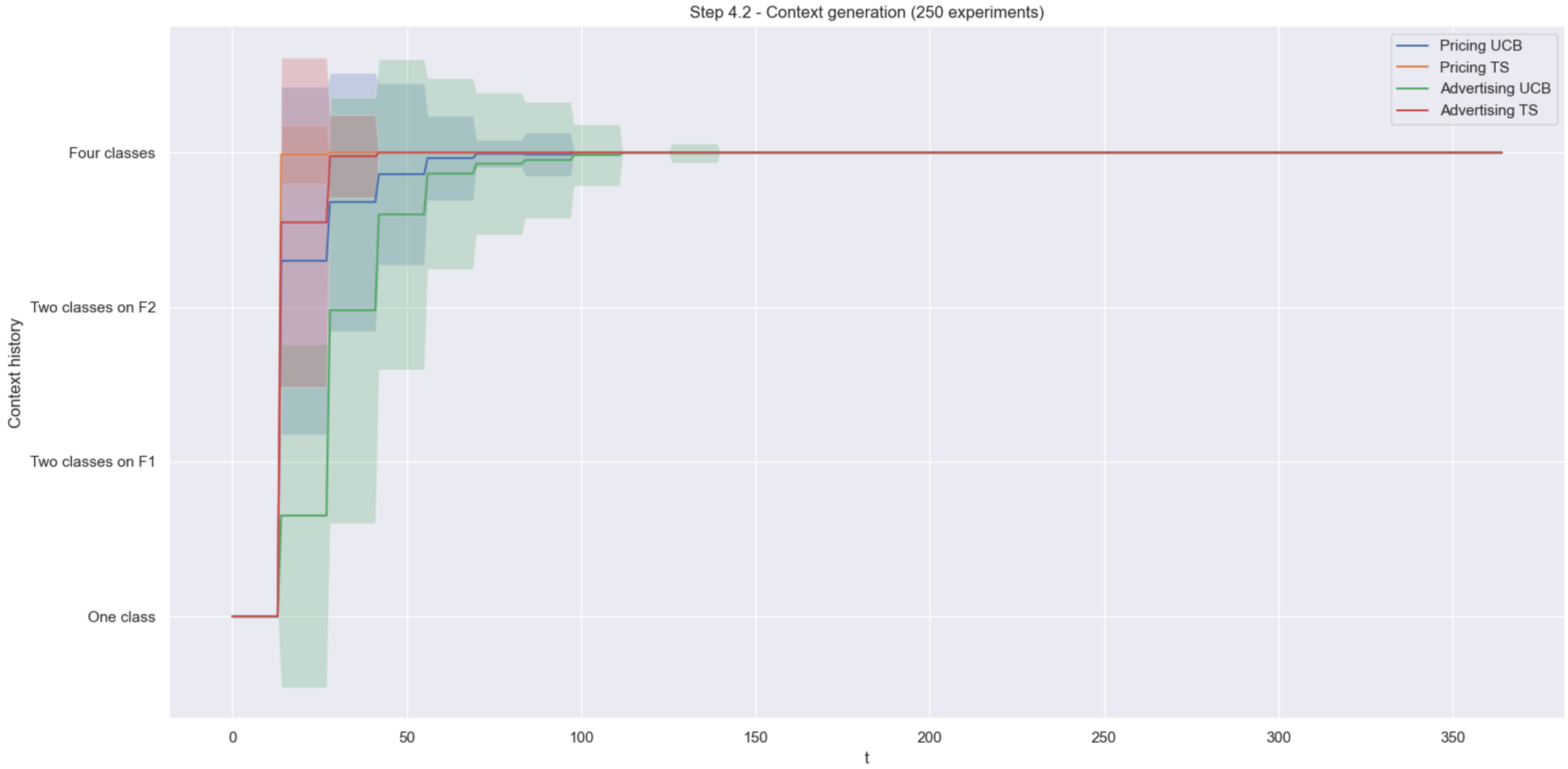

Step 4.1 - Context known (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Cumulative regret
t
500000
400000
300000
200000
100000
0
0
50
100
150
200
250
300
350

4.2 - Regret


Step 4.2 - Context generation (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Instantaneous regret
t
0
1000
2000
3000
4000
0
50
100
150
200
250
300
350

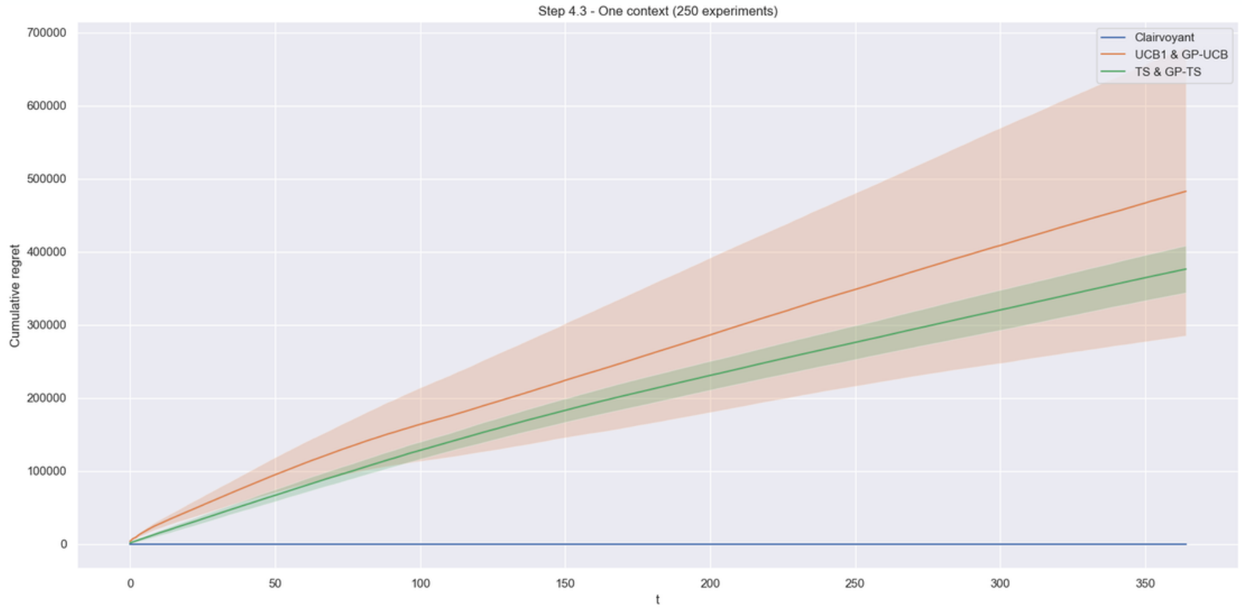

Step 4.2 - Context generation (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Cumulative regret
t
0
100000
200000
300000
400000
500000
0
50
100
150
200
250
300
350

4.2 - Choice of the Context

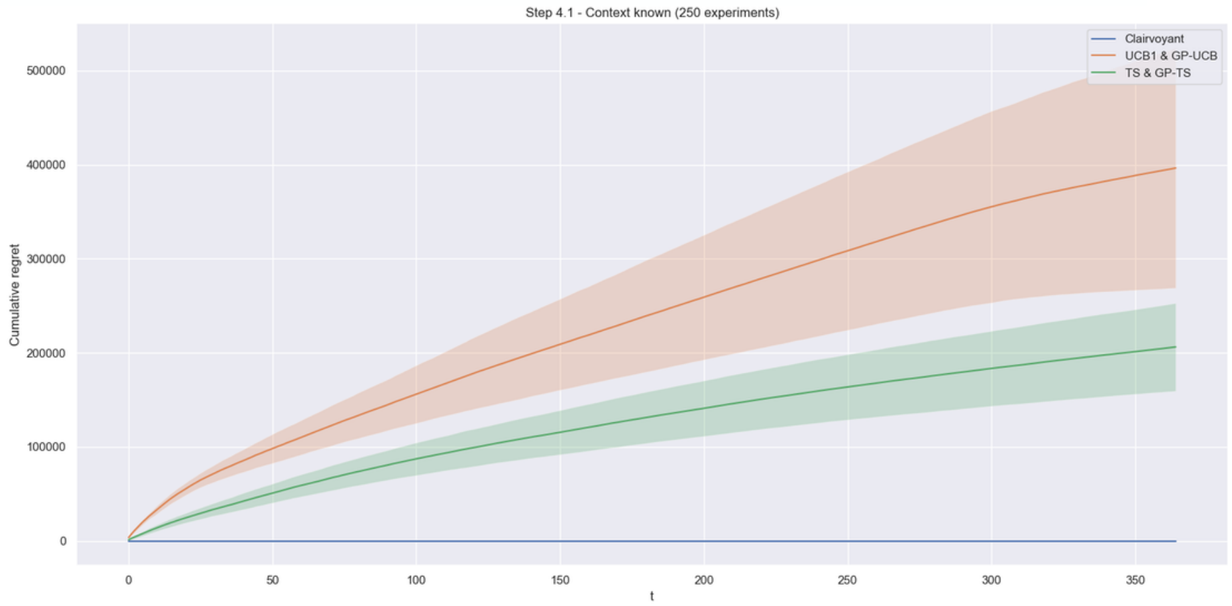


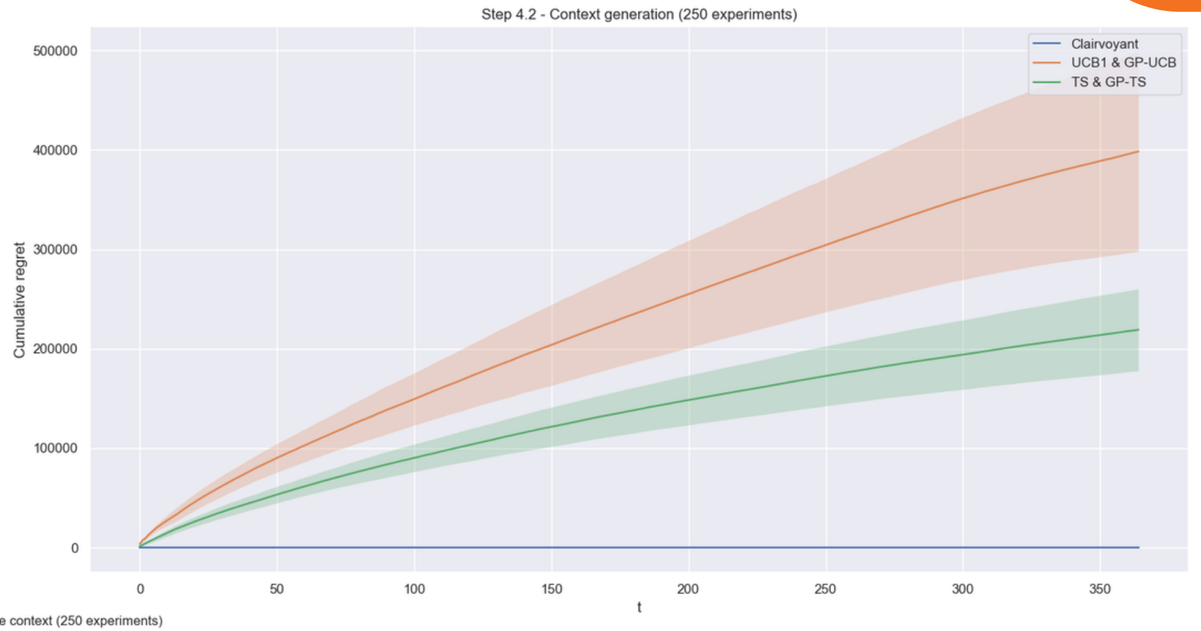
4.3 - Regret

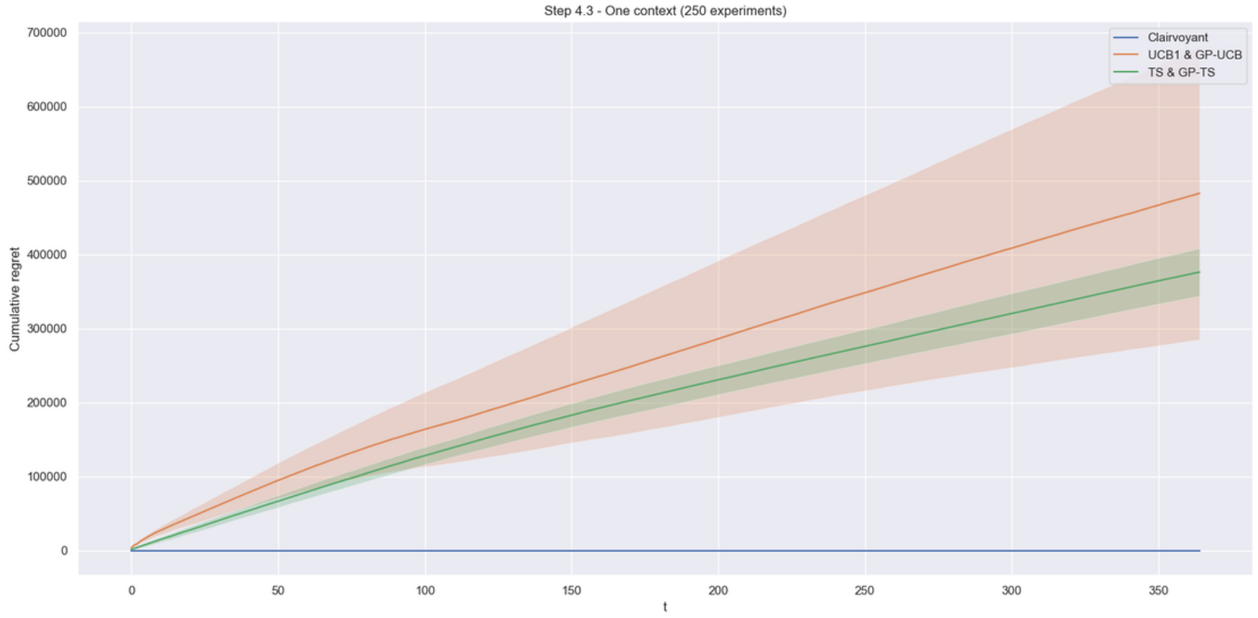

Step 4.3 - One context (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Instantaneous regret
t
0
1000
2000
3000
4000
0
50
100
150
200
250
300
350


Step 4.3 - One context (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Cumulative regret
t
0
100000
200000
300000
400000
500000
600000
700000
0
50
100
150
200
250
300
350

Comparison


Step 4.1 - Context known (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Cumulative regret
t
0
100000
200000
300000
400000
500000
0
50
100
150
200
250
300
350


Step 4.2 - Context generation (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Cumulative regret
t
0
100000
200000
300000
400000
500000
0
50
100
150
200
250
300
350


Step 4.3 - One context (250 experiments)
Clairvoyant
UCB1 & GP-UCB
TS & GP-TS
Cumulative regret
t
0
100000
200000
300000
400000
500000
600000
700000
0
50
100
150
200
250
300
350

Step 5: Dealing with non-stationary environments with two abrupt changes

Again we have a **single-user class** called C1.
We **know** the curves related to **advertising problems**, but we have **no prior information** about the **pricing curve**.
Additionally, the pricing curves are not constant; they change over time in a **non-stationary** manner, following **three distinct seasonal phases** throughout the time period.

The three seasonal phases can be considered the following:

1. **Winter:** In this period people travel less often than during the summer but more than in the spring. As a consequence, prices are **slightly lower** w.r.t. the summer season.
2. **Spring:** In this period people tend to not travel very often thus we expect to have **lower prices.**
3. **Summer:** This is the **holiday period** when people travel often hence prices tend to be **high.**

To address this situation, we will apply the UCB1 algorithm along with two variations of the UCB1 algorithm designed specifically for handling non-stationary data**.**

1. **Passive Non-Stationary UCB1:** using **Sliding Window**
2. **Active Non-Stationary UCB1:** using **Change Detection**

Non-stationary environment
Abrupt changes

Phases	Price1 - 15.50$	Price2 - 30.70$	Price3 - 60.20$	Price4 - 70.60$	Price5 - 90.80$
Winter	0.36	0.58	0.51	0.28	0.12
Spring	0.29	0.32	0.41	0.20	0.08
Summer	0.40	0.67	0.81	0.32	0.14

SW-UCB1

1. Play once every arm $a \in A$

2. At t , play an arm a_t with $n_{a_t}(t-1,\tau) = 0$ if any, otherwise play arm a_t such that

$$a_t \leftarrow \arg\max_{a \in A} \left\{ \bar{x}_{a,t.\tau} + \sqrt{\frac{2\ \log(t)}{n_a(t-1,\tau)}} \right\}$$

CD-UCB1

1. inItialize τ_a=0 for each arm $a \in A$
2. for each t

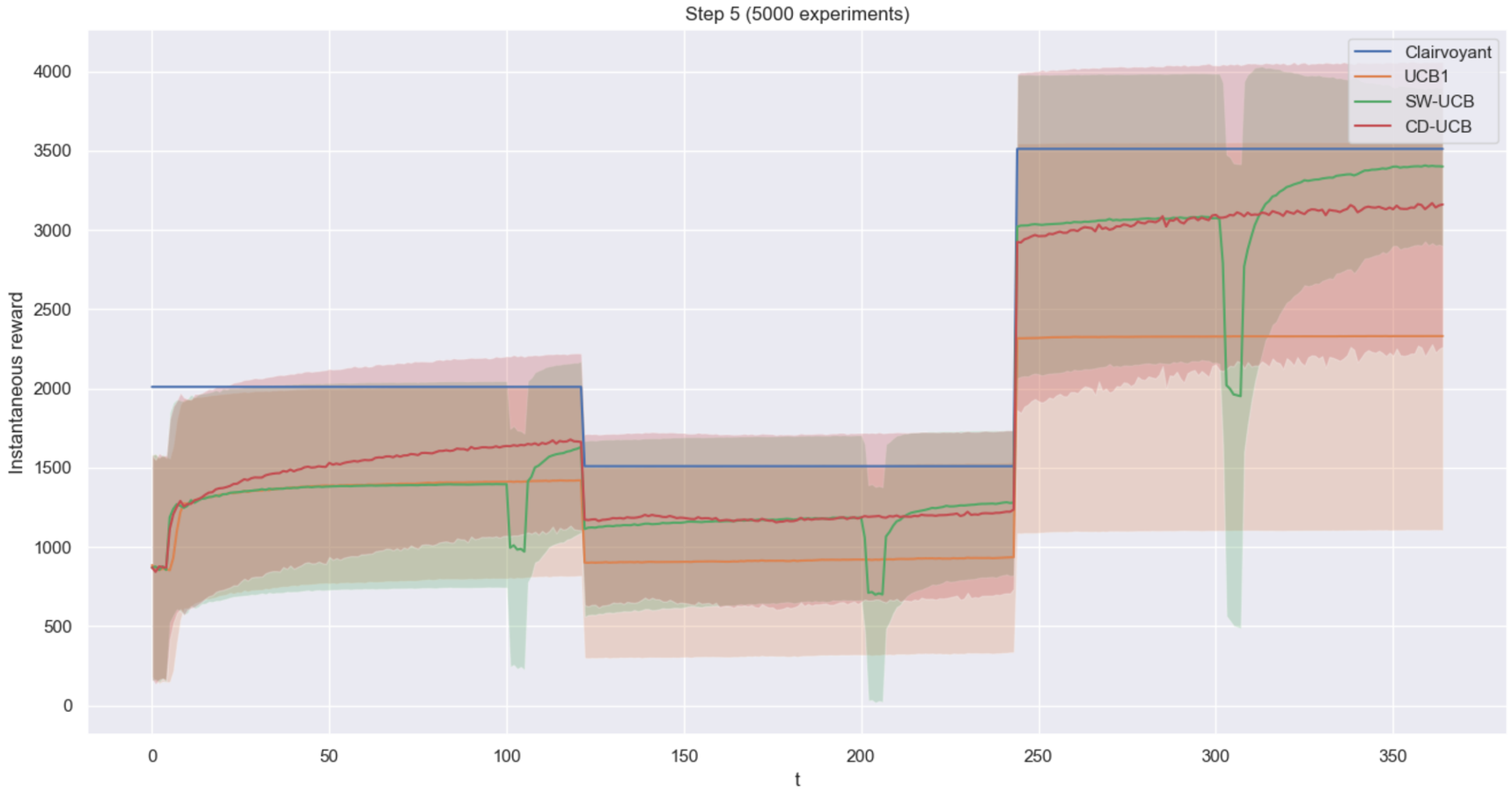
$$a_t \leftarrow \arg\max_{a \in A} \left\{ \bar{x}_{a,\tau_a,t} + \sqrt{\frac{2\log(n(t))}{n_a(\tau_a, t-1)}} \right\} \text{ with probability } 1-\alpha$$

$$a_t \leftarrow \text{random arm with probability } \alpha$$

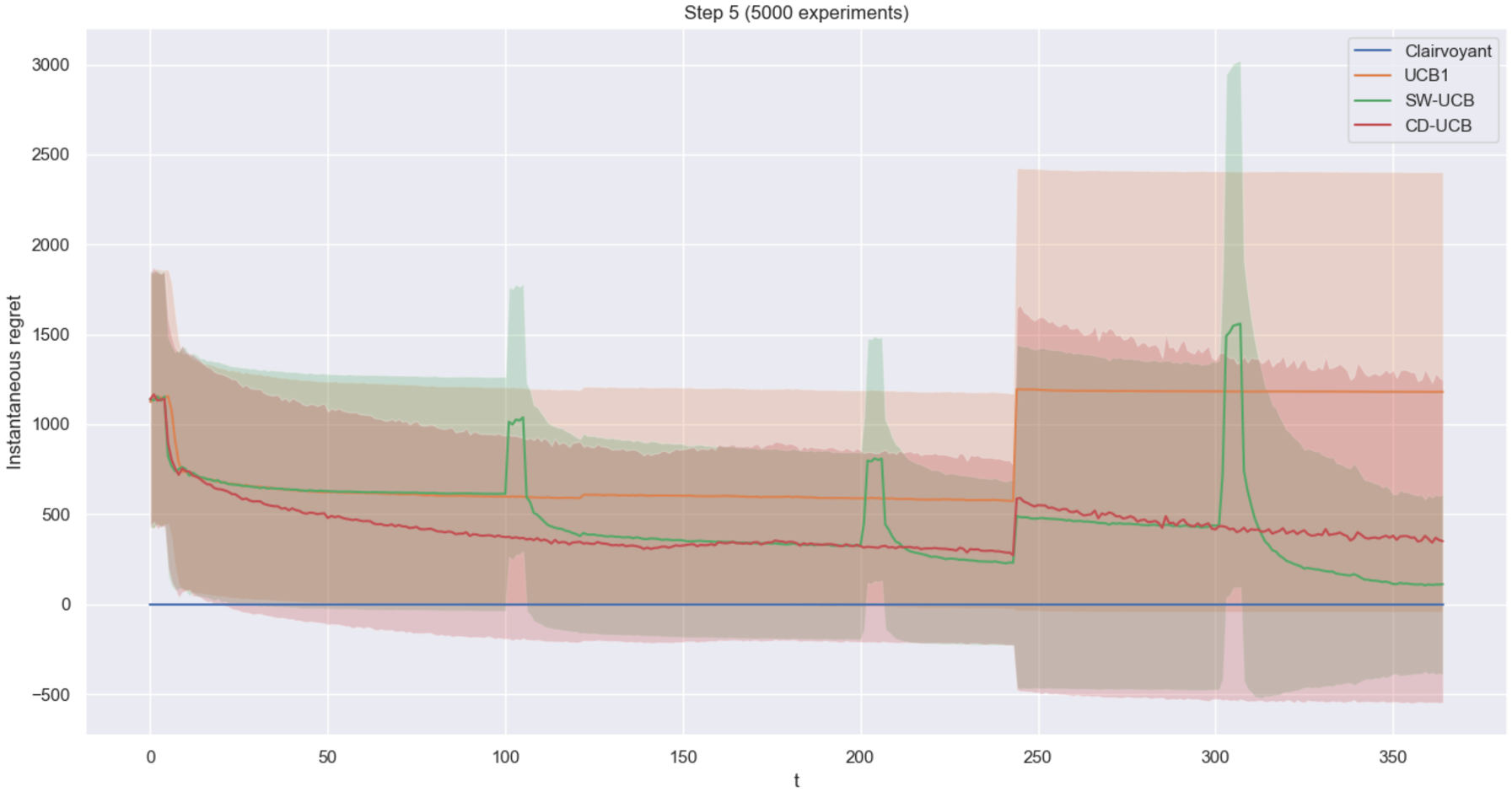
$n(t)$	is the total number of valid samples
$\bar{x}_{a,\tau_a,t}$	is the empirical mean of arm a over the last valid samples
$n_a(\tau_a, t-1)$	is the number of valid samples for arm a

3. collect reward r_t
4. if $CD_a(r_\tau, \ldots, r_t) = 1$ then $\tau_a = t$ and restart CD_a

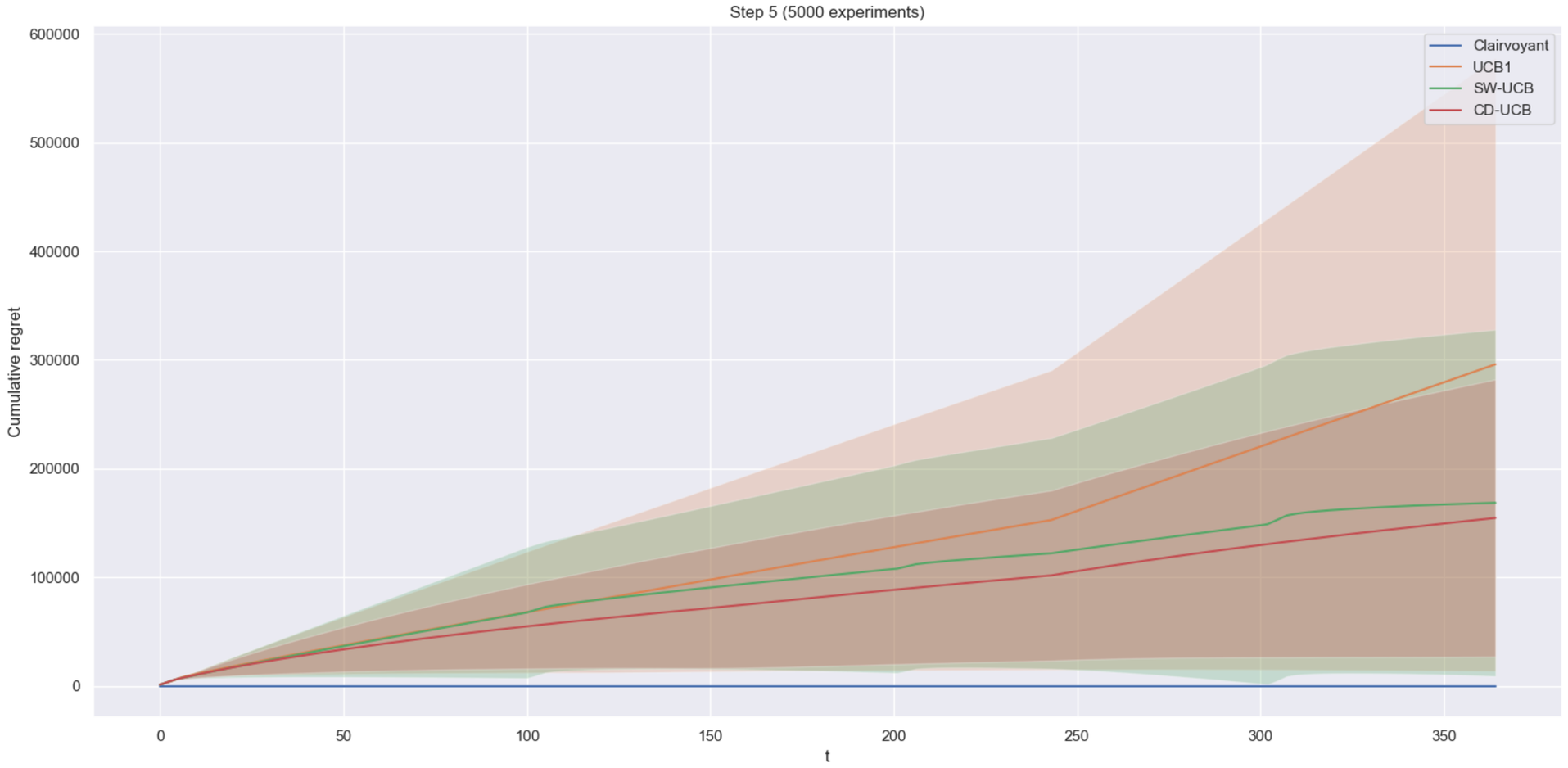
Instantaneous Reward



Instantaneous Regret



Cumulative Regret



Step 6: Dealing with non-stationary environments with many abrupt changes

First, in step 6.1, we add the **EXP3 algorithm** in the previous step scenario, considering a **fixed bid**. We expect that EXP3 will perform **worse** than the two non-stationary UCB1.

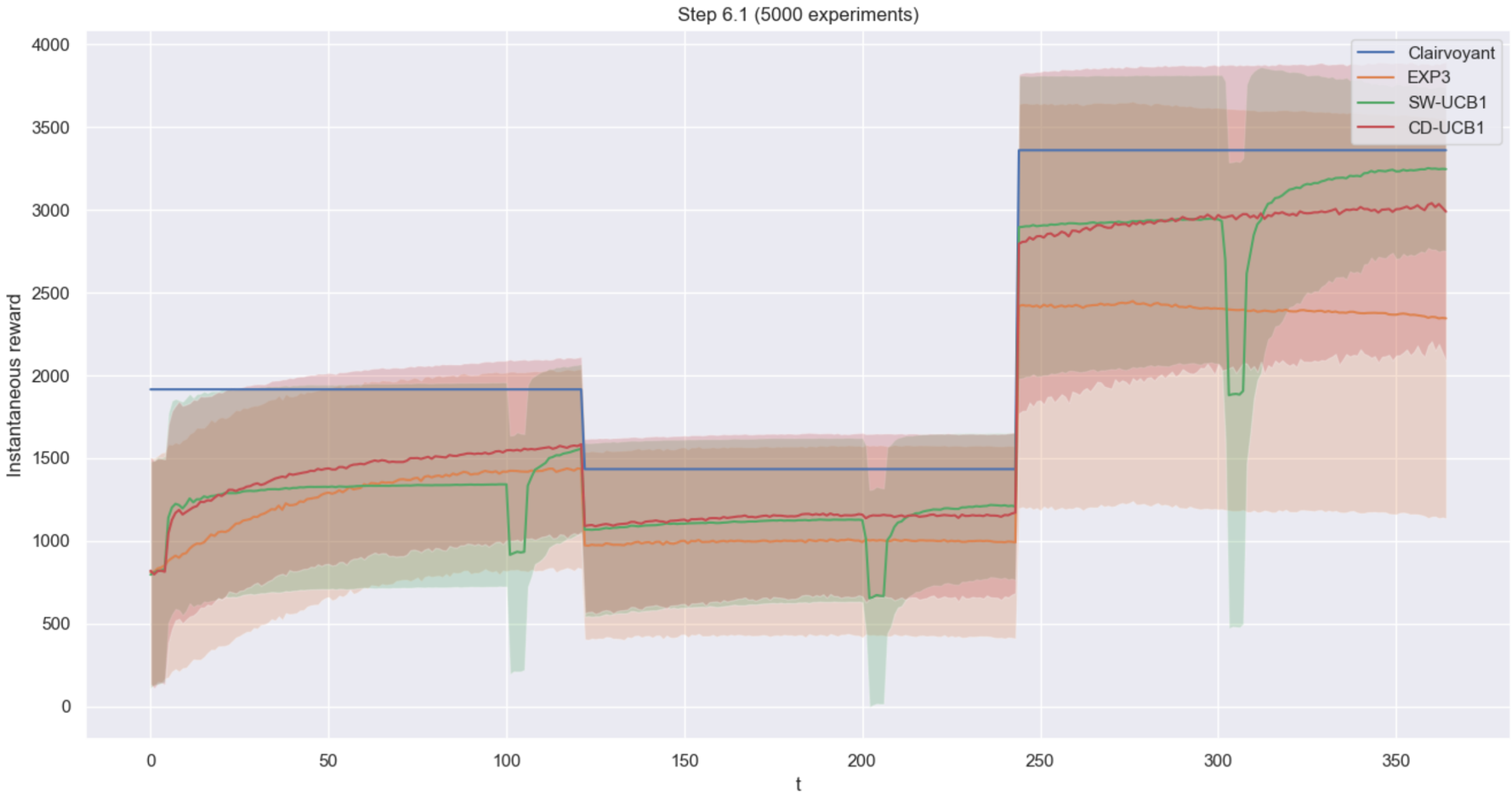
Next, in step 6.2, we'll consider a different non-stationary scenario characterized by a **higher degree of non-stationarity**. This increased rate of change is modelled by introducing **five phases**, each associated with a different optimal price.
These phases will **rapidly** and **cyclically** change.

In this new scenario, we'll apply the EXP3 algorithm, UCB1, SW-UCB1 and CD-UCB1. We expect that EXP3 will outperform the two non-stationary versions of UCB1.

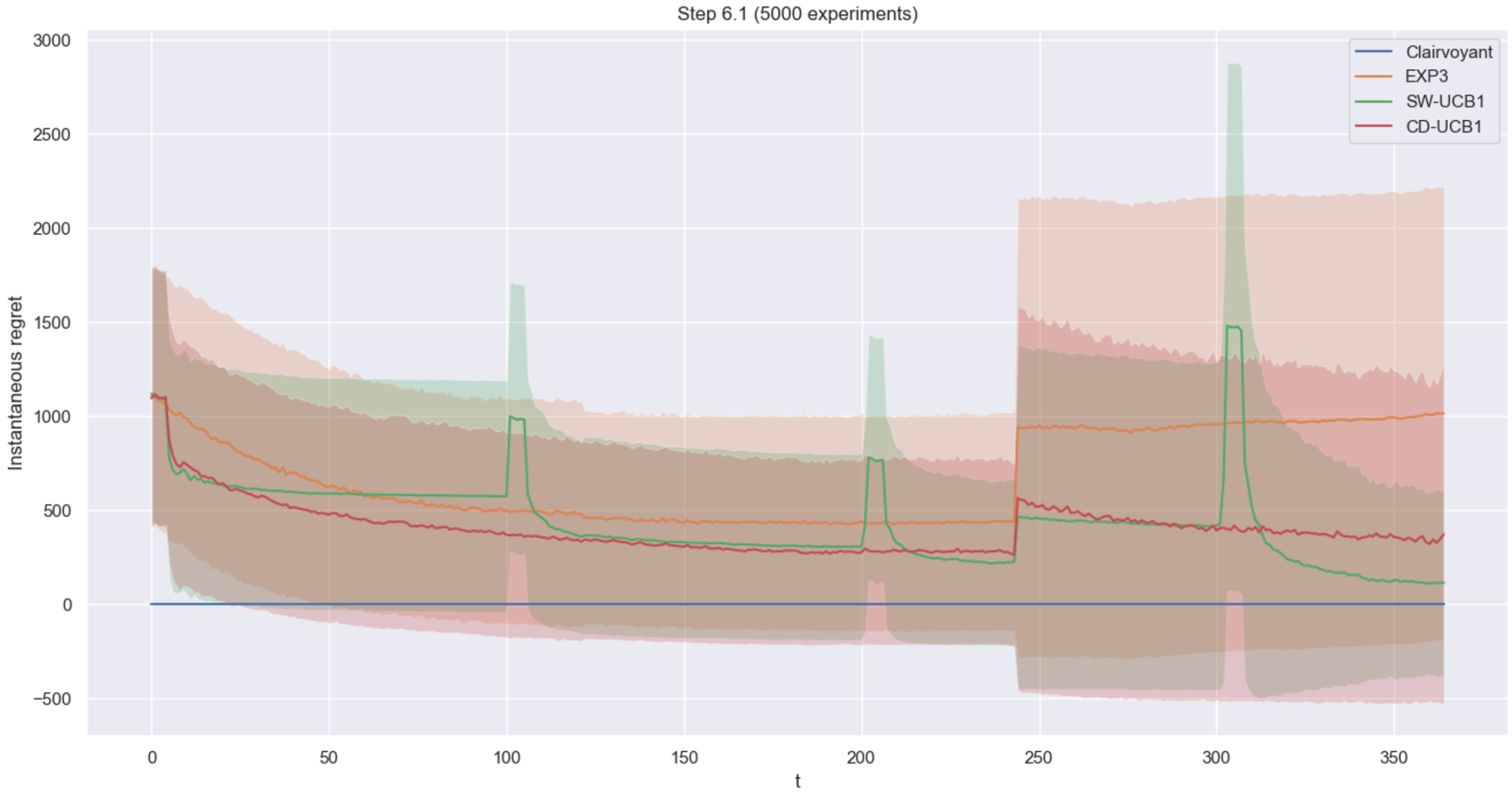
EXP3

1. Given $\gamma \in [0, 1]$, initialize the weights $w_i(1) = 1$ for $i = 1, \ldots, K$.
2. In each round t:
    1. Set $p_i(t) = (1 - \gamma)\dfrac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \dfrac{\gamma}{K}$ for each i.
    2. Draw the next action i_t randomly according to the distribution of $p_i(t)$.
    3. Observe reward $x_{i_t}(t)$.
    4. Define the estimated reward $\hat{x}_{i_t}(t)$ to be $x_{i_t}(t)/p_{i_t}(t)$.
    5. Set $w_{i_t}(t + 1) = w_{i_t}(t)e^{\gamma \hat{x}_{i_t}(t)/K}$
    6. Set all other $w_j(t + 1) = w_j(t)$.

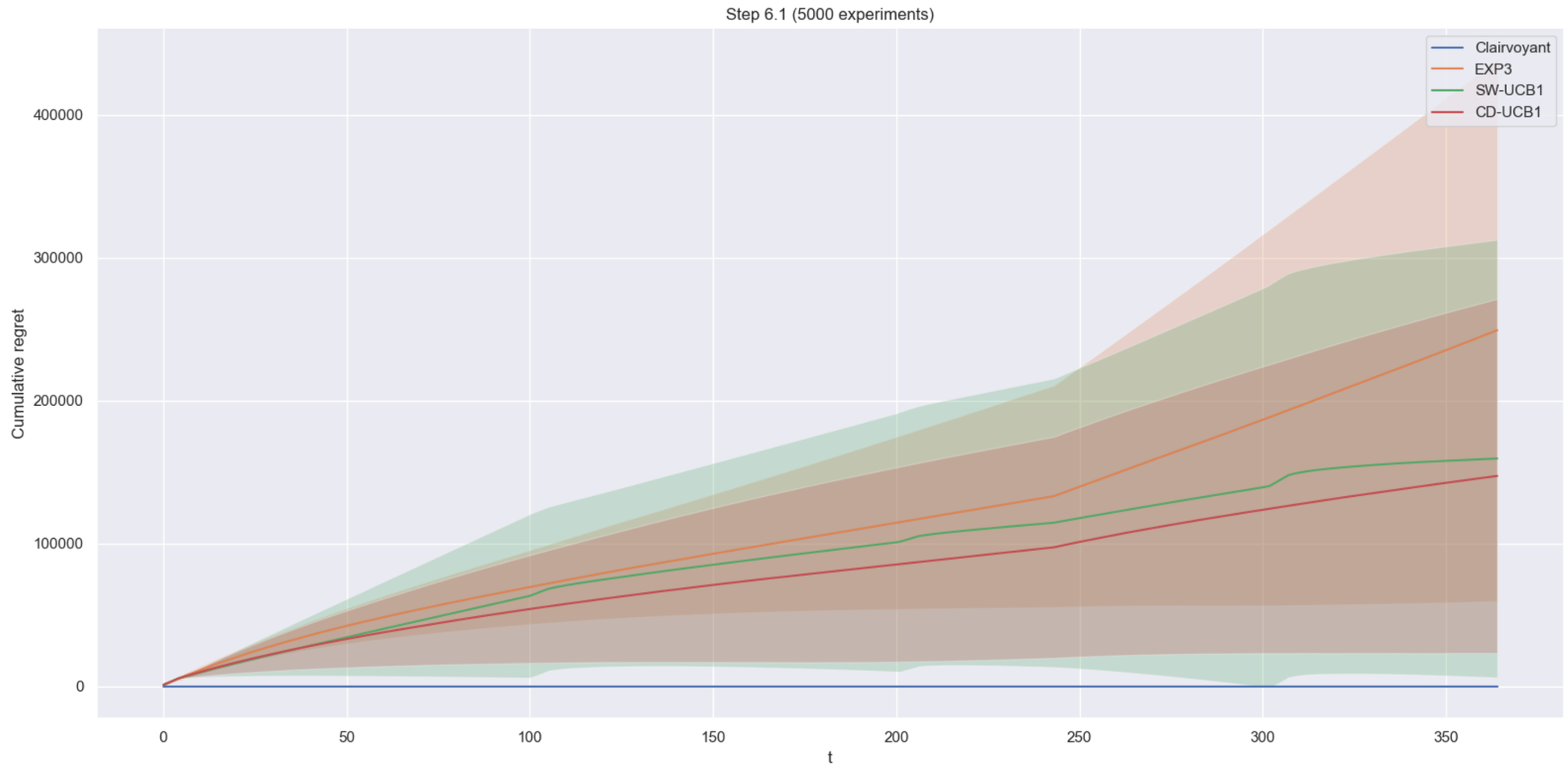
6.1 - Instantaneous Reward



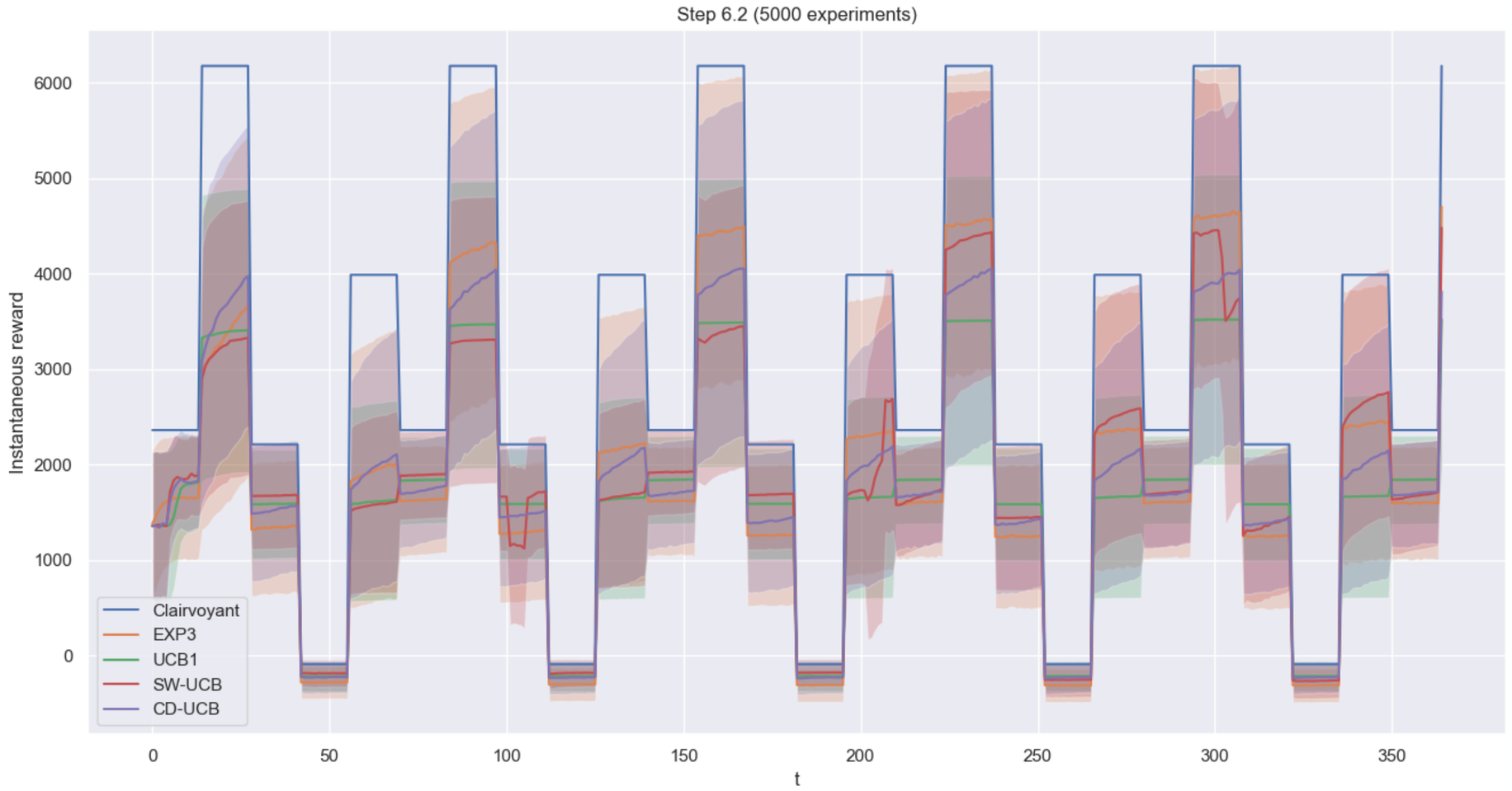
6.1 - Instantaneous Regret



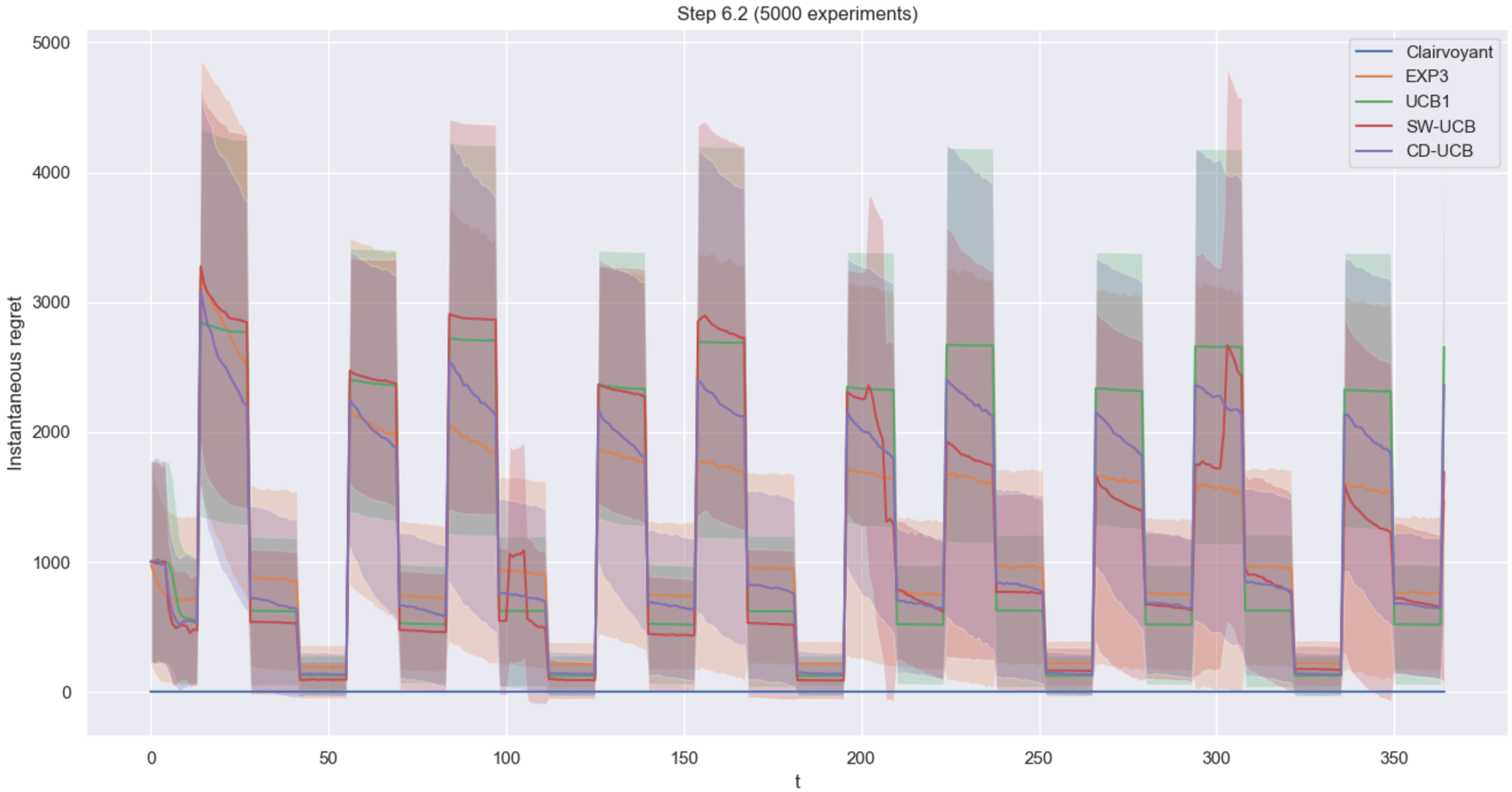
6.1 - Cumulative Regret



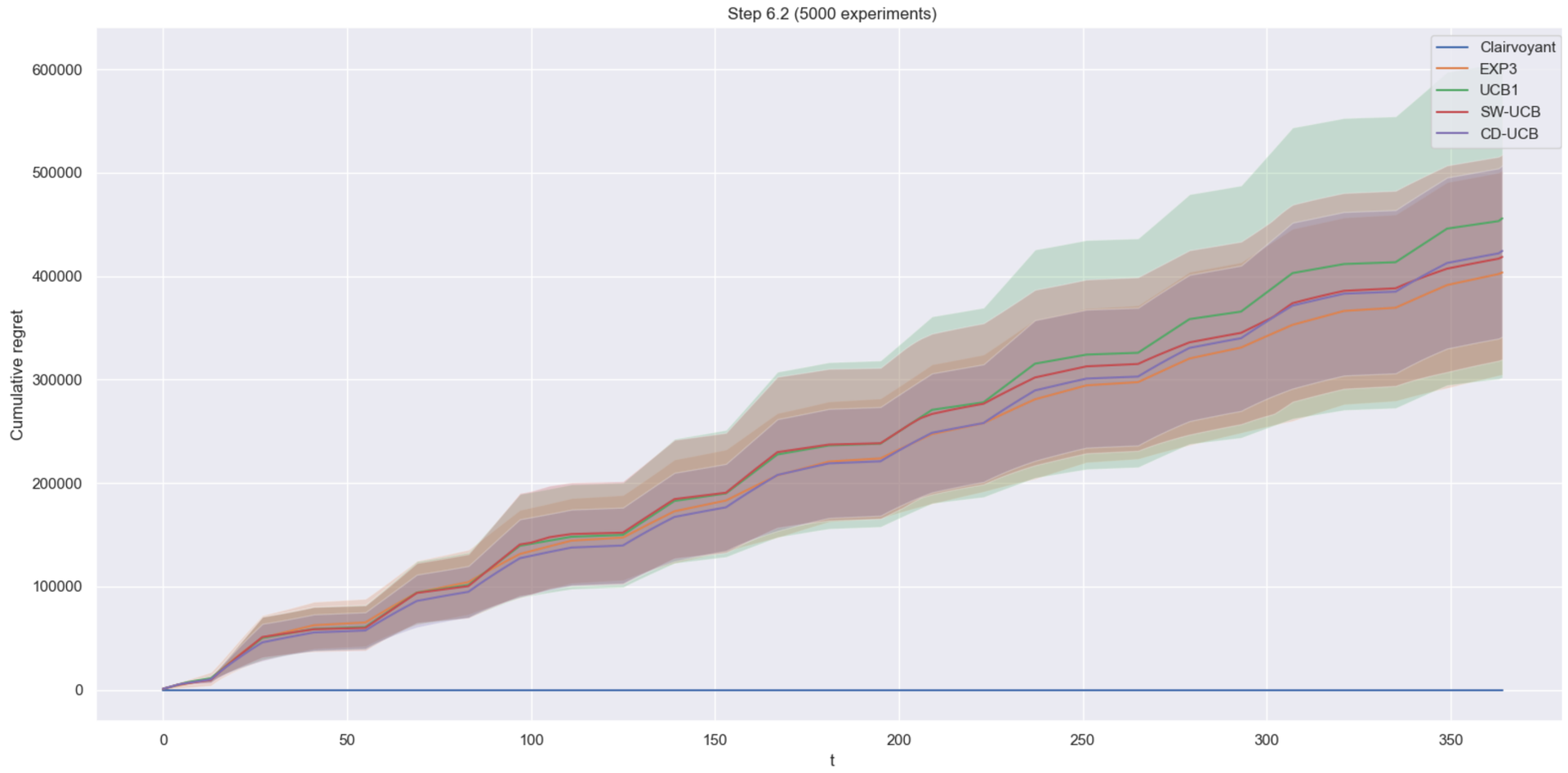
6.2 - Instantaneous Reward



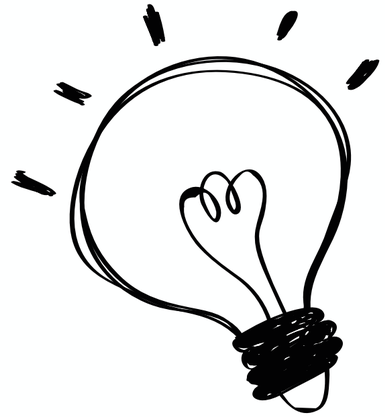
6.2 - Instantaneous Regret



6.2 - Cumulative Regret



Summary
6 Steps



01

Both algorithms have a **logarithmic** cumulative regret

02

Both algorithms have a **logarithmic** cumulative regret

03

The **cascade** application of the first two steps obtains **good** results

04

The case where the **context** is **known** is the **best**. The **context generation** obtains an **excellent** result.

05

SW and **especially CD** manage to make UCB **competitive** even in environments with **abrupt changes**

06

EXP3 outperforms every other algorithm with **very frequent scenario changes**

Improvements

01

Experimenting with **multiple** classes in every step

02

Experimenting with **unbalanced** classes

03

Experimenting with **adversial** environments

GOT QUESTIONS?

Reach out.

Project Github

POLITECNICO
MILANO 1863