Data Intelligence Applications

Pricing

Submitted By :
Matteo Biasielli
Emilio Capo
Stefano Cassini
Luca Dondoni

Contents

1 Preliminary Information

The full code is available here [1].

2 The Product

The product analyzed in this project is a pair of sunglasses. This is mainly due to three reasons:

- It is a product well known to the public;
- It is subject to seasonal variations of interest, as well as more subtle trend variations, which makes it an interesting product for our analysis;
- It both features aesthetic properties and material properties, thus it makes possible for different kinds of clients to be interested in it and to value it in different ways.

Figure 1: *The pair of sunglasses used.*

As for the product features, we defined:

- Materials: carbon fiber, aluminium;
- Lens: polarized, anti-reflex;
- Brand: no specification (in order to avoid both undervaluation, in case the product was perceived as "fake", and overvaluation, in case it was perceived as a luxury good).

[1] https://github.com/MatteoBiasielli/PricingInECommerce

3 Demands and Phases

3.1 Demands

To estimate the demand of the product, we decided to hand out a survey and gather the data ourselves. The survey is available here [2].
In structuring the survey, we brainstormed to define which the most useful information to gather would be, while still avoiding the collection of personal data and keeping the survey quick to fill in to encourage compilation and spreading. The latter was carried out through link sharing, so anyone who was provided with the related link could have access the survey.
The information we gathered is summarized in Table 1. It both includes basic information about people and their estimate of the maximum price they would spend for the product in object. This allowed us not only to estimate the aggregate demand curve, but also to explore what are the characteristics that may influence the maximum price that people assign to this product.

[2] https://docs.google.com/forms/d/e/1FAIpQLSexdzjGGr2XVAPcFaYI_Js7j_dONkfoeWPCjwLXST2JxVlTrw/viewform?usp=sf_link

	Domain (Sample Coverage)
Gender	Male (32.5%), Female (67.5%)
Age [15÷72]	15÷19 (11.6%), 20÷24 (57.1%), 25÷29 (13.1%), 30÷39 (5.5%), 40÷49 (3.3%), 50÷59 (7.3%), 60÷72 (2.1%)
Continent	Southern Europe (85.4%), Western Europe(2.7%), Northern Europe (0%), Eastern Europe (4.3%), North America (0.6%), South America (0.9%), Asia (5.2%), Africa (0.9%), Oceania (0%)
Studied after high school	Yes (79.3%), No (20.7%)
Educational background	Artistic (8.5%), Humanistic (36.5%), Scientific (34.3%), None (20.7%)
Employed	Yes (36.2%), No (63.8%)
Spends time outdoor	Yes (47.4%), No (52.6%)
Maximum price [€ 0÷450]	0÷30 (10.9%), 31÷60 (20.1%), 61÷90 (12.5%), 91÷120 (17.9%), 121÷150 (15.8%), 151÷180 (2.4%), 181÷210 (9.4%), 211÷240 (0.9%), 241÷270 (3.7%), 271÷300 (4.6%), 301÷330 (0%), 331÷360 (0%), 361÷390 (0%), 391÷420 (0.9%), 421÷450 (0.9%)

Table 1: *Table showing the information gathered for each data point (330 data points in total). Though sample coverage for fields "Age" and "Maximum price" is aggregated in ranges, this is only to provide a meaningful representation of the coverage distribution. Each data point has two single non-negative integers representing age and maximum price. The educational background was only observed from people who continued their studies after high school.*

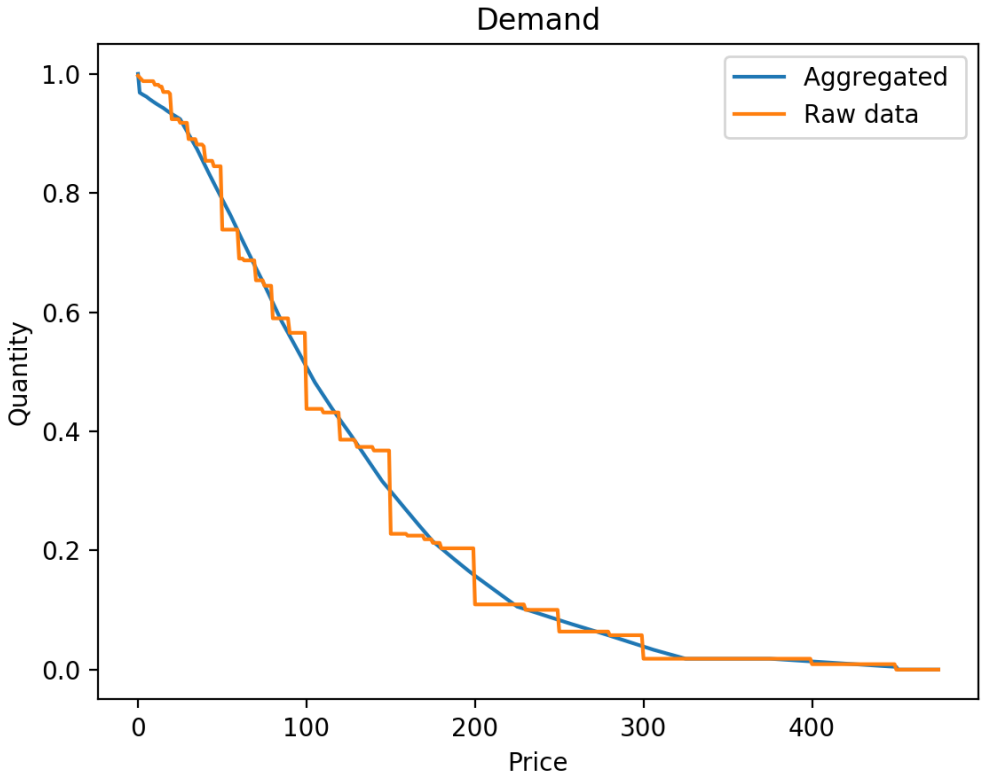

Figure 2: *Resulting aggregated curve.*

We were, then, able to find three significant classes of people:

- people that are **employed** and **live in Southern Europe**;
- people that are **not employed** and **live in Southern Europe**;
- people that **do not live in Southern Europe**.

In Fig. 3, we show the aggregated demand curve and the three disaggregate curves in the phase of high interest. The value "1" on the y-axis represents the highest possible quantity (e.g. all users) and "0" represents "no user". This figure has the only purpose of showing the four curves together in order to highlight the differences among them. In fact, a more detailed description of all the curves during the different phases is given in Section 3.2

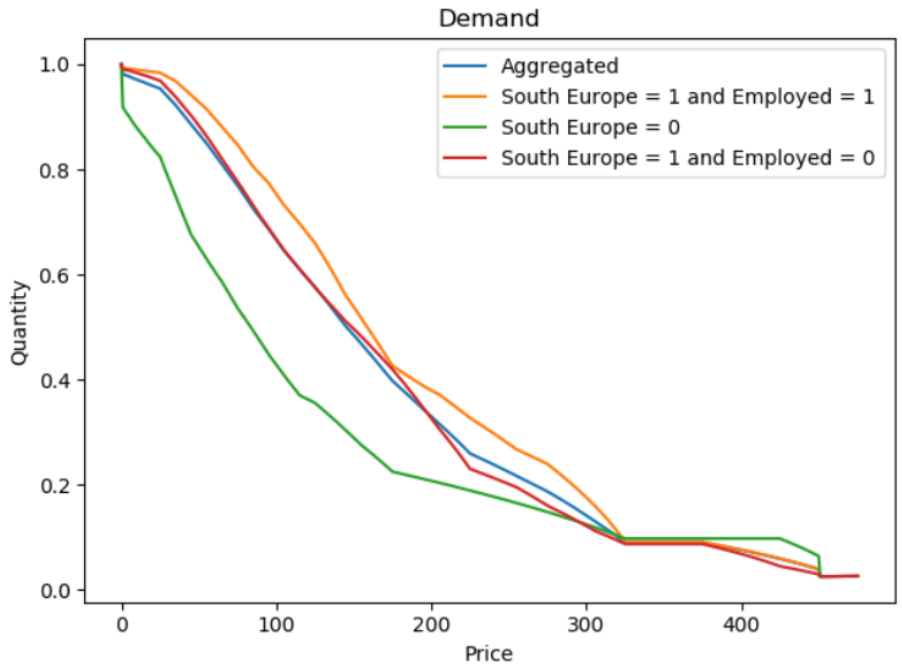

Figure 3: *Aggregate and Disaggregate Demand Curves in the Phase of High Interest.*

3.2 Phases

Figure 4: *People's interest for sunglasses over the past year from Google Trends.*

The definition of the phases was guided by Google Trends, which gives a precise idea of what are the periods of the year in which a certain keyword is searched more often.
Fig. 4 and 5 illustrate very clearly the periodic pattern that people's interest for the keyword **Sunglasses** follows. In the figures, the value "100" on the y-axis represents the moment of maximum interest, while the value "0" represents no interest. As a simplifying assumption, we divided the year in three macroperiods, which are:

- **Phase of high interest**: March to July.

Figure 5: *People's interest for sunglasses over the past five years from Google Trends*

- **Phase of medium interest**: August to November.
- **Phase of low interest**: December to February.

Following, Fig. 6, 7, 8 and 9 show both the aggregate curve and all the disaggregate curves in the three phases.

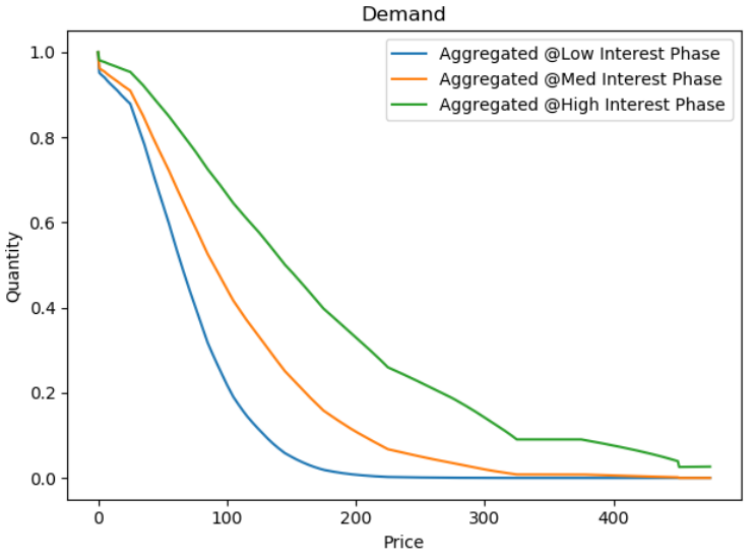

Figure 6: *Aggregate Demand Curve in the three phases.*

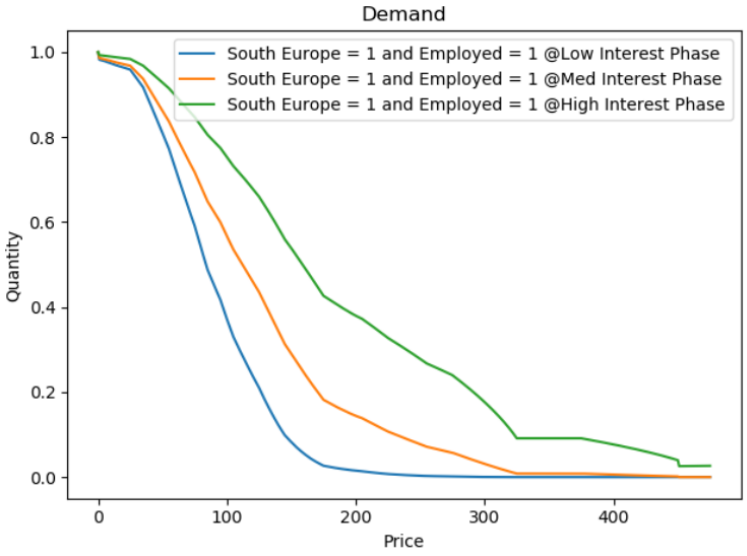

Figure 7: *Demand Curve for the category of people that are employed and live in Southern Europe in the three phases.*

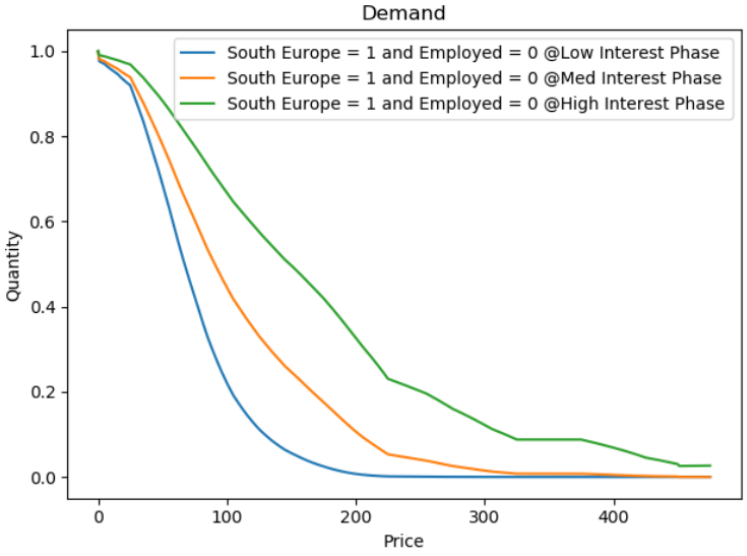

Figure 8: *Demand Curve for the category of people that are not employed and live in Southern Europe in the three phases.*

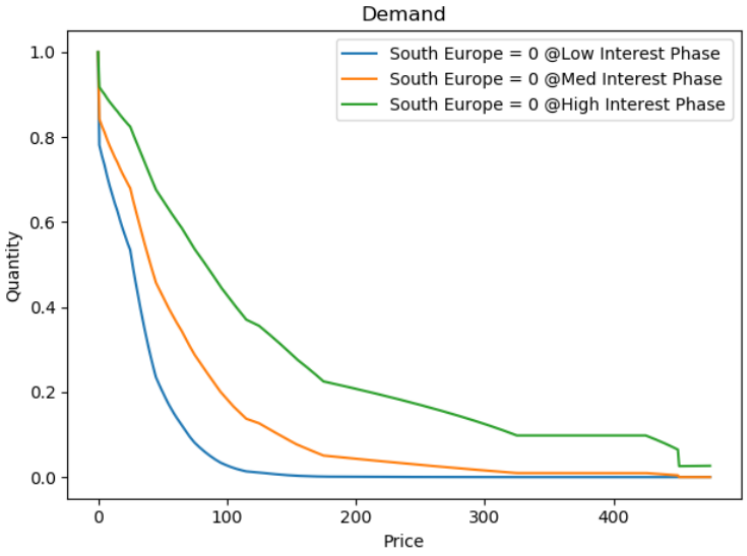

Figure 9: *Demand Curve for the category of people that do not live in Southern Europe in the three phases.*

4 Experimental Setup

The very first step of the process is the definition of the size of the experiments. In particular, assuming that we know how much one pair of sunglasses costs to the online shop (including manufacturing, transportation and/or shipping, etc.), we need to define:

- the **number of arms**; the higher it is, the more likely is the chance of exploring arms that are close to the optimal arm. On the other hand, having more arms implies that the *optimization horizon* will be required to be longer to allow a proper exploration of all the arms.
- the **optimization horizon**; this is the amount of time that will be spent running the price optimization algorithms. The longer this is, the more are the possibilities that can be explored. However, exploring more possibilities causes to have a bigger loss - or *regret* - in the initial phases of the exploration.

It appears clear that the the two quantities we have to define depend on each other and represent the key of the optimization. A reasonable **optimization horizon** is **one year**. In fact, as described in Section 3.1 and Section 3.2:

- the interest that people have for our product behaves in the same way every year (Fig. 5);
- the phases alternate cyclically and the period of the cycle is **one year** (Fig. 4).

Thus, as long as we are able to collect enough data during one year, it makes no sense to extend the *optimization horizon* to a time that is longer than this.

From now on, we will assume to know that the **total cost** of a pair of sunglasses is 10€ for the shop and that the website of the online shop receives - by logged in users - **50 visits per day**.
It follows that during one year the shop will collect $50*365 = 18250$ data points. The number of time steps that our optimization algorithms will take is, then, $T = 18250$. Consequently, this defines the maximum number of arms that we can effectively explore, that are:

$$\lceil \sqrt[4]{T * logT} \rceil = 17$$

Being the cost for the shop of a pair of sunglasses equal to 10€, we will - for obvious reasons - not consider arms with a price that is lower than this. We want the choice of the arms to be independent from the data we collected using the survey, as if we really knew nothing about the demands, so we simply picked the arms uniformly:

starting from 30€, one arm every $(370€ - 30€)/n$, where n is the number of arms and is between 1 and 17. By picking the arms this way, the lowest arm is always $a_1 = 30€$ and the highest one is always $a_n = 350€$. a_1 is the price that could be given to a very cheap pair of sunglasses, while a_n represents the other, expensive, extreme, so we expect the optimal price to lie somewhere in the middle of the two. Formally, we can define all the arms as:

$$a_i = 30€ + (i-1) * (370€ - 30€)/n \quad \forall i \in [1, 17]$$

The values of the candidates, as well as the corresponding Bernoulli expected values and marginal profits are reported in Table 5 (the bold row represents the optimal arm). The table has the only purpose of numerically exemplifying what is showed in Figure 3.

4-Arms Learner (high interest phase)			
Arm Number	**Candidate Price**	**Bernoulli Exp. V.**	**Marginal Profit**
1	30	0.938	18.76 €
2	**137**	**0.531**	**67.54 €**
3	244	0.233	54.60 €
4	350	0.090	30.76 €

Table 2: *Values of the 4 candidates derived from the aggregate demand curve for the high interest phase. The optimal arm is highlighted in bold.*

5-Arms Learner (high interest phase)			
Arm Number	**Candidate Price**	**Bernoulli Exp. V.**	**Marginal Profit**
1	30	0.938	18.76 €
2	110	0.627	62.79 €
3	**190**	**0.357**	**64.32 €**
4	270	0.195	50.73 €
5	350	0.090	30.76 €

Table 3: *Values of the 5 candidates derived from the aggregate demand curve for the high interest phase. The optimal arm is highlighted in bold.*

10-Arms Learner (high interest phase)			
Arm Number	**Candidate Price**	**Bernoulli Exp. V.**	**Marginal Profit**
1	30	0.938	18.76 €
2	66	0.805	45.10 €
3	102	0.657	60.52 €
4	**138**	**0.528**	**67.59 €**
5	174	0.401	65.84 €
6	210	0.303	60.62 €
7	246	0.230	54.38 €
8	282	0.175	47.73 €
9	318	0.106	32.67 €
10	350	0.090	30.76 €

Table 4: *Values of the 10 candidates derived from the aggregate demand curve for the high interest phase. The optimal arm is highlighted in bold.*

17-Arms Learner (high interest phase)			
Arm Number	**Candidate Price**	**Bernoulli Exp. V.**	**Marginal Profit**
1	30	0.938	18.76€
2	50	0.868	34.71€
3	70	0.789	47.35€
4	90	0.706	56.50€
5	110	0.628	62.79€
6	130	0.558	66.97€
7	**150**	**0.485**	**67.89€**
8	170	0.416	66.52€
9	190	0.357	64.32€
10	210	0.303	60.62€
11	230	0.253	55.66€
12	250	0.225	53.90€
13	270	0.195	50.73€
14	290	0.161	45.12€
15	310	0.122	36.65€
16	330	0.090	28.95€
17	350	0.090	30.76€

Table 5: *Values of the 17 candidates derived from the aggregate demand curve for the high interest phase. The optimal arm is highlighted in bold.*

4-Arms Learner - Optimal arm in the 3 phases			
Phase ID	**Optimal Arm**	**Optimal Candidate**	**Marginal Profit**
Low	1	30 €	16.72 €
Med	2	137 €	35.92 €
High	2	137 €	67.54 €

Table 6: *Data of the optimal arms of the 4-arms learner in the 3 different phases of the environment. It's important to notice that, even though the 4-arms learner is the most promising learner in case of stationary environment, in this new king of scenario, in particular in phases 'low' and 'med', it provides the worst performances if compared with the 17-arms and 10-arms learners.*

10-Arms Learner - Optimal arm in the 3 phases			
Phase ID	**Optimal Arm**	**Optimal Candidate**	**Marginal Profit**
Low	2	66 €	27.32 €
Med	3	102 €	39.81 €
High	4	138 €	67.59 €

Table 7: *Data of the optimal arms of the 10-arms learner in the 3 different phases of the environment.*

17-Arms Learner - Optimal arm in the 3 phases			
Phase ID	**Optimal Arm**	**Optimal Candidate**	**Marginal Profit**
Low	3	70 €	27.02 €
Med	4	90 €	39.91 €
High	7	150 €	67.89 €

Table 8: *Data of the optimal arms of the 17-arms learner in the 3 different phases of the environment.*

Additionally, in the case of non-stationary environment, it is important to define a proper value for the sliding window, that is, the length of the buffer where the latest collected samples are stored. The different formulas we considered to compute a reasonable sliding window length, as well as their pros and cons, are reported below:

- Environment information are not known:

$$SW_1 = \sqrt{T}$$

The basic formula for the computation of the window length. It only considers the number of interactions T our algorithm is going to perform with the environment.
Pros: Very simple;
Cons: With large numbers of arms it leads to bad performances;

$$SW_2 = N * \sqrt{T}$$

It introduces a dependency in the number of arms N, that we assume linear;
Pros: It's very simple and overcomes the problem of playing with a large number of arms;
Cons: Since the linear dependency in the number of arms is an our hypothesis, it may exists another dependency, non-linear, better than this one that we don't know.

- Environment information are known a priori (unlikely to happen in practice):

$$SW_3 = 2 * B * \frac{1}{F} * \sum_{f=1}^{F} (Rew_{max,f} - Rew_{min,f}) * \sqrt{\frac{T * \log T}{F - 1}}$$

This formula is a slight modification of the one discussed in the paper "*On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems*" by Aurélien Garivier and Eric Moulines. It takes into account both the maximum reward in phase "f" $Rew_{max,f}$ and the minimum reward in phase "f" $Rew_{min,f}$ (since we're doing profit maximization, this two values will correspond respectively to the maximum marginal profit and the minimum marginal profit). It considers the number of phases F of the demand curve (in our case $F = 3$) as well;

Pros: It takes into account various characteristics of the environment;
Cons: It does not consider the time instants at which the abrupt changes take place. Furthermore, we need to know a lot of information about the environment;

$$SW_4 = \frac{T}{F}$$

Provided that all the phases of the demand curve have the same length (and therefore they are uniformly distributed over the time horizon), with this formula we're setting the sliding window size equal to this length, in order to start exploring the "old" arms in correspondence of the beginning of a new phase.
Pros: Very simple and tailored for the specific case;
Cons: The phases should be uniformly distributed over the time horizon and their lengths have to be equal.

5 Optimization on Aggregate Demand

5.1 Introduction

Before discussing the performances of each algorithm, we define a tool that will be useful in the analysis, the Gain function (or G-function), as:

$$G_{L1,L2}(k) = CR_{L1}(k) - CR_{L2}(k)$$

The function $CR_L(k)$ quantifies the cumulative reward, given by a learner L, at interaction k. In other words, how much we gained up to interaction k by optimizing with learner L.
The function $G_{L1,L2}(k)$ quantifies the difference between how much we gained with learner *L1* and how much we gained with learner *L2*, up to interaction k.
Since the values of k are discrete, the values that the function $G_{L1,L2}(k)$ will assume are discrete as well. For the analysis that is following, we are going to assume a linear interpolation for the values of this function between different ks, in order to assure the continuity of the $G_{L1,L2}(k)$ function and, therefore, the existence of the derivative w.r.t. k in every specific point k'.

The function $G_{L1,L2}(k)$ typically shows 3 different behaviours:

1. The first derivative, computed in the specific point *k'*, is negative (i.e. the function is decreasing in that point). This means that, at that interaction, we are gaining less by optimizing with learner *L1* rather than with learner *L2*. In formulas:
$$\frac{d}{dk} * G_{L1,L2}(k)\Big|_{k'} < 0$$

2. The first derivative, computed in the specific point *k'*, is zero. This means that, at that interaction, what we gained with the learner *L1* is the same as what we gained with the learner *L2*. In formulas:
$$\frac{d}{dk} * G_{L1,L2}(k)\Big|_{k'} = 0$$

3. The first derivative, computed in the specific point *k'*, is positive (i.e. the function is increasing in that point). This means that, at that interaction, we are gaining more by optimizing with learner *L1* rather than with learner *L2*. In formulas:
$$\frac{d}{dk} * G_{L1,L2}(k)\Big|_{k'} > 0$$

5.2 K-Testing

The purpose of this section is to highlight the behaviour of the k-testing algorithm in the pricing domain defined earlier. Obviously, since k-testing assumes a **stationary environment**, during each experiment we will consider a fixed demand curve that does not change with time.
The algorithm was implemented following a **sequential** approach that consists of several **a/b tests** (depending on the number of candidates chosen). During each phase a variation candidate is tested against the control one and, if successful, it becomes the new control candidate. In the case we cannot prove that the variation is better with statistical significance, the control candidate remains the same as before and the variation candidate is discarded. This procedure is applied until all candidates have been tested and the last control is chosen as a winner.
By behaving this way, **at the end of each experiment**, we are always guaranteed a solution that represents the **candidate to be used during the exploitation phase**. Furthermore, to reinforce the assumption of not knowing the best candidate a priori, at the beginning of the algorithm the candidates are shuffled and consequently they are tested against each other in a random fashion. This, combined with the fact that after each hypothesis test the samples are discarded, ensures that each candidate is treated equally.
Before running the algorithm, it's important to discuss how we chose the number of samples to be used in the following case studies.

Time Horizon

There are two options when selecting the time horizon in which the k-testing algorithm should run:

1. The first option is to utilize the statistical tools in order to determine the minimum number of samples needed to obtain a satisfying result, given certain precision parameters. This is what we ideally want to do so that we may achieve some guarantees on the effectiveness of the algorithm. The formula to obtain the minimum number of samples needed for each candidate per hypothesis test is the following:

$$n = \frac{(z_{(1-\alpha)} + z_{\beta})^2 \sigma^2}{\delta^2}$$

Where:
α = Type I error, β = Type II error σ = Standard deviation,
δ = Minimum variation in the process,
$z_{(q)}$ = Value of the Gaussian distribution for quantile q.

2. The second option is tho fix the time horizon (and consequently the available number of samples) prior to choosing all the other parameters. This is a more restrictive condition and is applied in practice when we don't know the real variance of the process or we have a limited time available to perform our analysis.

In the first case, **the resulting time horizon changes depending on some statistical parameters** (alpha, beta, delta), on the variance of the data **but also on the number of candidates** we decide to compare. In fact, increasing the number of candidates leads to the collection of more samples, these are in turn used to perform all the hypothesis tests needed to declare a winner. More specifically, each run of the algorithm is composed exactly of n-1 phases (where n is the number of candidates) in which two candidates are compared and where we always need to collect the same fixed number of samples for both contenders. This method is impractical if we want to compare the performances of the algorithm since also the time horizon changes in each configuration. Besides that, we are required to know the actual variance of the process in order to apply the formula for the minimum needed samples.

In the second approach instead, we **fix the time horizon in advance** so that we always know when the algorithm is going to end. In theory we would like to have the number of samples as high as possible so to estimate better the expected reward of each candidate. In practice this decision is influenced by the resources available to the company given that the optimization process may cause financial losses in the short term. In any case, having a fixed time horizon allows us to discard the assumption of knowing the variance of the data, which is a more realistic expectation. Furthermore, this way we can **easily compare the performance of the algorithm** by running it in parallel with other ones or with different parameter configurations. Since the time is fixed, we expect that the results obtained will be straightforward to confront.

In the following experiments **we will use both these approaches** depending on the aspects to be highlighted.

First Example

For the moment, let's assume to know the variance of the process and to have unlimited resources, so that we can utilize the formula for the minimum number of samples defined previously.
If we consider the case of 17 candidates (as seen in Section 4), alpha = 0.05, beta

= 0.1 and delta = 1 we have that in the **phase of high interest** the minimum number of samples to collect for each candidate (during each test) is 1402. This amounts to a total of $16 * (1402 * 2) = 44864$ data points to be collected during the exploration phase of the algorithm. In addition to those samples, we collected further 4000 rewards for the exploitation phase to show the change in rewards of the two stages.

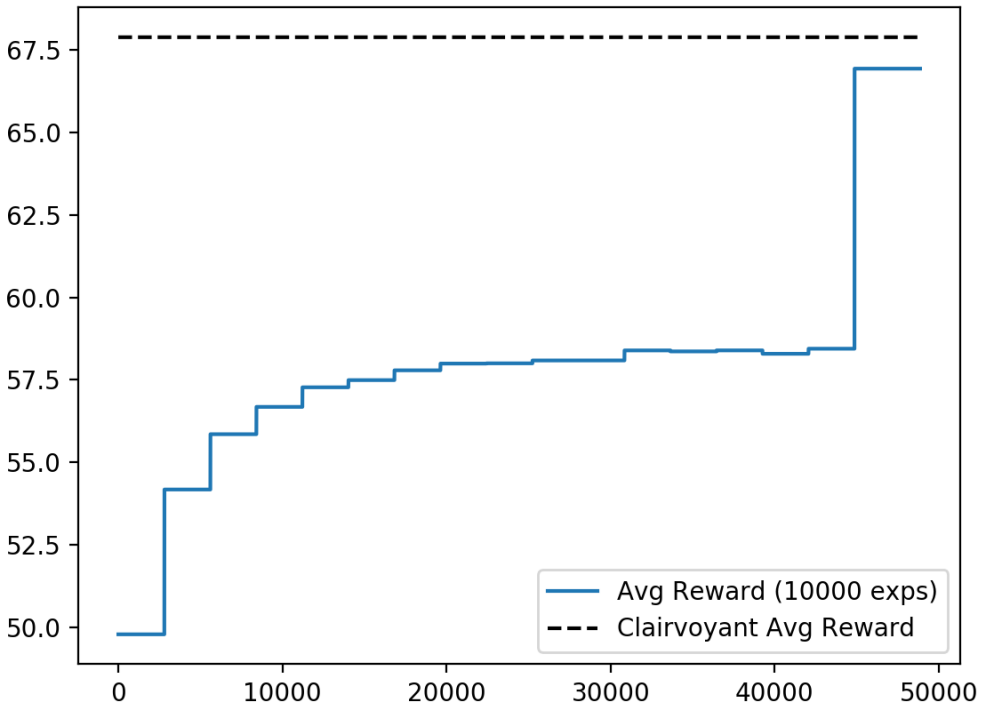

Figure 10: *Expected reward during the exploration phase and the beginning of the exploitation one. This plot averages 10000 experiments in the described scenario.*

As we can see from the plot, there is a clear difference between the two phases of the algorithm.
In the **exploration phase** the reward slowly climbs up by growing with increasingly small steps towards the optimal one. Each step represents a stage in which two candidates are tested one against each other, samples are collected for both candidates and then an hypothesis test is applied to declare the best one. We can notice that, as the time progresses, the expected reward of each of these steps tends to increase. This is due to the fact that, test after test, the best candidates are the one that

carry on towards the end of the phase and therefore they contribute positively to the average reward collected.
In the **exploitation phase** we can observe a big jump in expected reward since the best candidate (as found by the algorithm) is selected and then used exclusively to collect the future rewards. It can be noted that, in average, the candidate selected as the final one is not so far off the optimal (given a sufficient number of samples) and thus the final average reward will be usually quite high.

In this specific example we observed the following winning candidates out of the 10000 experiments conducted:

```
Total wins by candidate: [0, 0, 0, 0, 98, 2943, 3800, 2288, 845, 26, 0, 0, 0, 0, 0, 0, 0]
```

As we can see, the algorithm **was able to identify the best arm 3800 times out of 10000** experiments and the majority of errors were attributed to the two nearest candidates (6th and 8th), which have really close values to the optimal (see Table 5 in Section 4). Overall the candidate that won the most was still the optimal one, this is why we can see in the reward plot a final value that is pretty close to the clairvoyant expected reward.

Despite a somewhat good final outcome, **we must take a look at the regret collected by the algorithm during the exploration**, in order to have a better understanding of its performance.
In Figure 11 we can see that, right away, k-testing starts collecting an high amount of regret. This a consequence of the fact that initially the rewards received are, in average, fairly low (first big steps in Figure 10). After that, for the majority of the exploration phase, the curve seems to lower slightly but we can see that overall the cumulative regret still shows an almost linear increase. This phenomenon is caused by the corresponding expected rewards stabilizing to, some extent, as the time increases. Finally, at the beginning of the exploitation phase, we observe how the curve almost flattens out towards zero regret. As we said before, since the final candidate is on average the optimal one or a slightly sub optimal one, we will have that in the future the regret collected will be reasonably low.
In conclusion, we can say that the good long term performance of k-testing unfortunately does not come for free. In fact, the **big amount of regret collected in the short term horizon** could have a big impact as loss in profits by the company. This may or may not be a good reason to choose an alternative optimization algorithm for the pricing scenario.

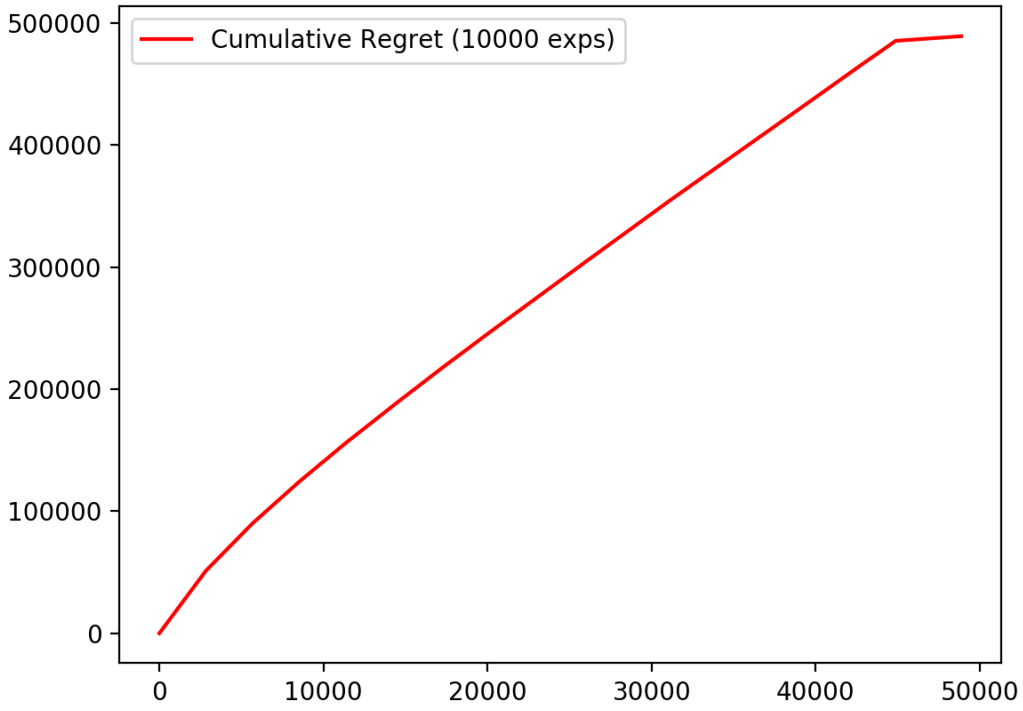

Figure 11: *Cumulative regret collected by the algorithm during the exploration phase and the beginning of the exploitation one. This plot averages 10000 experiments in the described scenario.*

Changing The Value Of Alpha (Type I Error)

The parameter alpha plays an important role during the execution of the k-testing algorithm. In fact, it represents the Type I error, which is the probability to reject the null hypothesis when it is true. Since each a/b test implies an hypothesis test, changing alpha has a big impact on the candidates selected during the process. If we choose a **small alpha**, we ask the algorithm to be **really precise** and to state that the variation candidate has an higher mean with respect to the control only when really confident. On the contrary, if we choose an **high value for alpha** we **relax the precision** of the test and take more "risks", so to speak. As always, there is not a best way to find the perfect value but we are most likely to find out that it's usually neither too small nor too high.

For the following experiment we take into account the **demand in its high interest phase**. In this particular case **we fixed the time horizon** to be at most equal to 18250. This allows us, as stated previously, to easily compare the performance of the algorithm with the variation of alpha. Indeed, a different alpha value would change the minimum number of samples needed for each configuration to be compared. In addition to being more realistic, this scenario can also show us how the k-testing algorithm behaves in the real time horizon of the pricing problem.

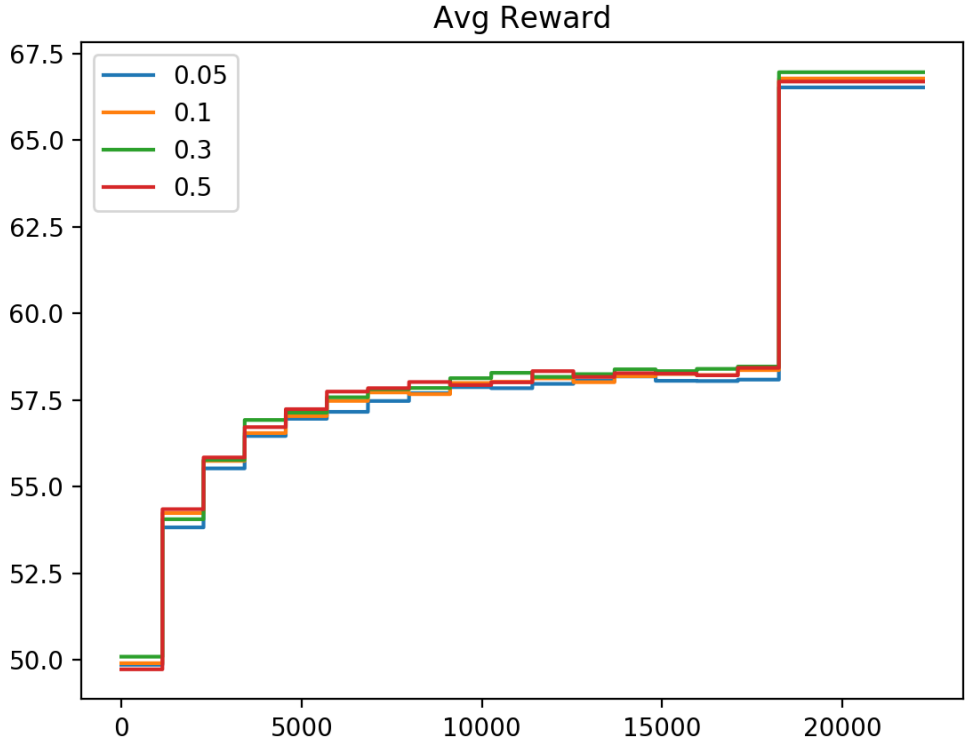

Figure 12: *Expected reward of the algorithm with different values of alpha (with 17 candidates). This plot averages 10000 experiments for each curve in the described scenario.*

In Figure 12, we plotted the expected reward for four different values of alpha (0.05, 0.1, 0.3, 0.5) and fixed the candidates to 17. We instantly notice that, during the exploration phase, the curves are really close one another. This is to be expected since we know that in this stage the candidates are confronted in a random order. In fact, the slightly different selected candidates between configurations are somewhat averaged out with the random opponents they face each time (given that we have pretty close values of alpha). As we can see in the plot, at the end of the time horizon the configuration with **alpha = 0.3 is the best one overall**. The total wins by candidates are summarized below (remember that the optimal candidate is the 7th one):

```
With 17 candidates, 0.05 alpha:
    Real values:  see Table 1 in Section 2
    Total wins by candidate: [0, 0, 0, 0, 458, 2661, 3346, 2154, 1126, 247, 6, 2, 0, 0, 0, 0, 0]
With 17 candidates, 0.1 alpha:
    Real values:  see Table 1 in Section 2
    Total wins by candidate: [0, 0, 0, 0, 266, 2793, 3660, 2237, 947, 96, 1, 0, 0, 0, 0, 0, 0]
With 17 candidates, 0.3 alpha:
    Real values:  see Table 1 in Section 2
    Total wins by candidate: [0, 0, 0, 4, 176, 2800, 4154, 2189, 596, 75, 3, 3, 0, 0, 0, 0, 0]
With 17 candidates, 0.5 alpha:
    Real values:  see Table 1 in Section 2
    Total wins by candidate: [0, 0, 0, 8, 323, 2624, 3925, 2083, 782, 204, 28, 18, 5, 0, 0, 0, 0]
```

If we look at either Figure 12 or the table just presented, we notice that there seems to be a trend that relates the value of alpha to the performance of the algorithm. As alpha initially increases (0.05, 0.1, 0.3) we constantly get better final rewards, yet this is no longer true when we set it equal to 0.5. What happens is that, **for relatively small values of alpha, we lose some good potential candidates** because we cannot prove they are a better alternative given the samples collected (that in this case are quite a small number). Loosening this constraint helps rejecting less alternative candidates, therefore it helps avoiding this problem to some extent. However, **if we set alpha too high, we risk adopting sub-optimal candidates** because we are less precise with our analysis.

In the plot shown below in Figure 13 we can see that the cumulative regret behaves accordingly to what we said so far. One thing that can be noted is that, for the realistic time horizon of 18250 samples, the **k-testing algorithm collects about 200000 of regret on average**.

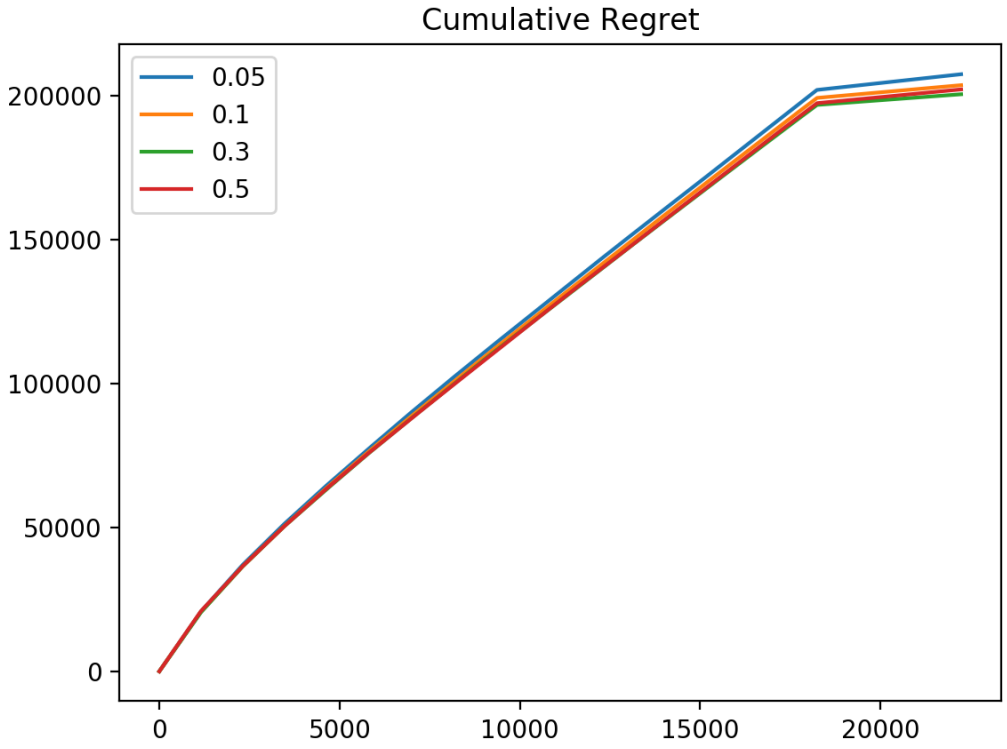

Figure 13: *Cumulative regret of the algorithm with different values of alpha (with 17 candidates). This plot averages 10000 experiments for each curve in the described scenario.*

Different Number Of Candidates

Every time we run the algorithm, we face a crucial step in which we must decide the number of candidates to be analyzed and their respective values. **This decision greatly affects the short and long term rewards collected by the algorithm and, consequently, the overall regret gathered throughout the process**. As a company, it's important to understand the advantages and disadvantages gained by choosing a different configuration.
So far we have only seen the case of 17 candidates, which was the upper bound of arms computed in Section 4. In the following practical example we will explore various configurations of arms and analyze the changing behaviour in order to highlight some general concepts regarding the choice of candidates.

For this scenario we considered the demand curve in its high interest phase, alpha = 0.3, beta = 0.1 and delta = 1. In order to compute the number of samples needed for each collection stage, we used the formula defined in the Time Horizon subsection. As we will see, the length of the "steps" in each configuration is almost identical but we can still notice that it differs slightly. This is due to the fact that the only aspect changing between configurations that affects the formula is, in fact, the variance of the arms we select.
We chose to plot the performance of the algorithm with 4, 5, 10 and 17 arms all selected uniformly in the interval $30 \div 350$ and whose values can be found in Table 2, 3, 4 and 5 respectively.

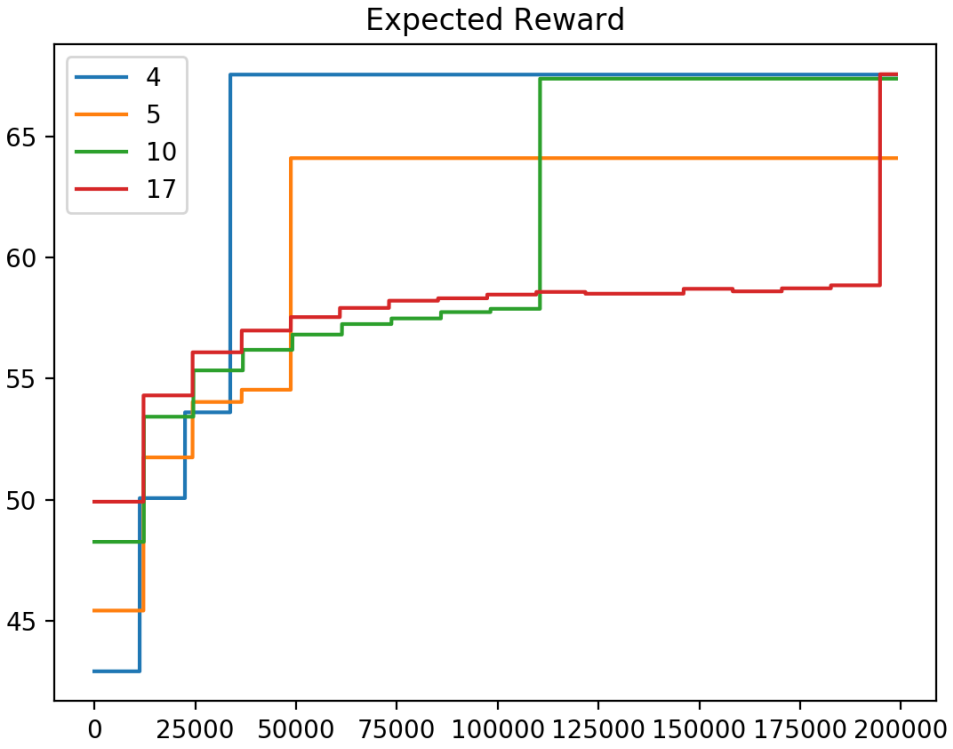

Figure 14: *Expected reward of the algorithm with different numbers of candidates selected uniformly in the interval* $30 \div 350$ *(with alpha= 0.3). This plot averages 10000 experiments for each curve in the described scenario.*

If we look at Figure 14 and Figure 15, we notice right away some interesting behaviours regarding the convergence of the different learners. For the moment let's not consider the learner with 4 arms (blue curve) since it's a special case that will

be discussed later.
As expected from the theory, we see that, as the number of arms increases, the length (in time) of the exploration phase is incremented. Since this value is fixed before running the algorithm, we know that we will **need to collect a big amount of samples if we have a lot of candidates to analyze**. As a consequence, we can see that the configurations with a smaller quantity of arms quickly move towards their optimal value (needing only a small number of samples) and thus collect less regret in the short time horizon.

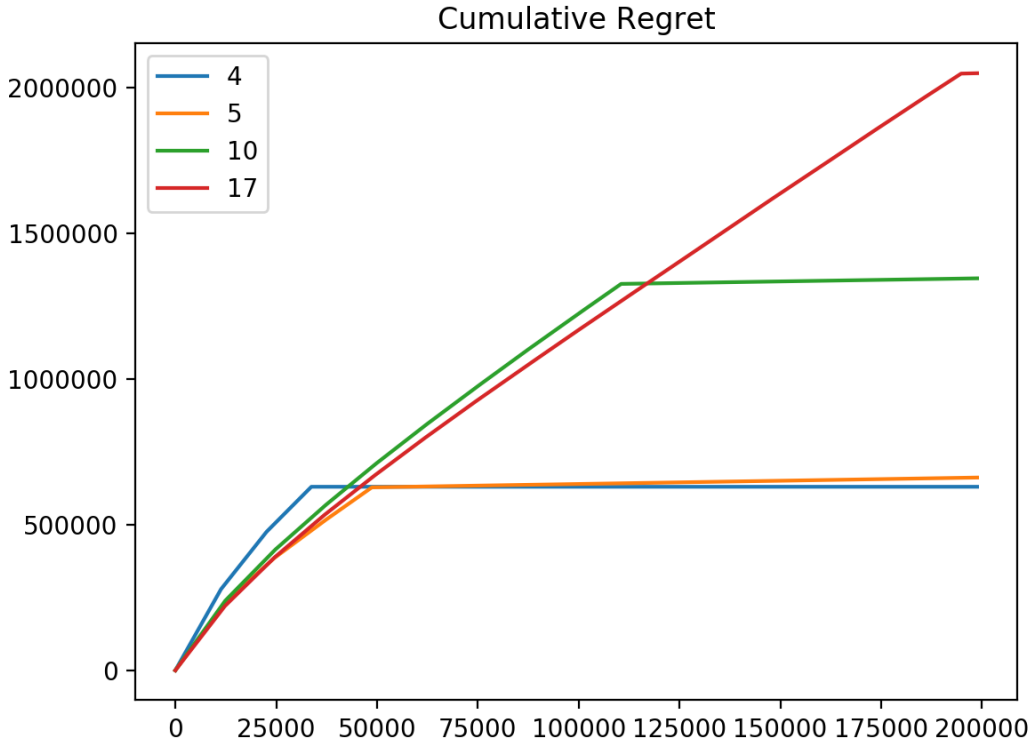

Figure 15: *Cumulative regret collected by the algorithm with different numbers of candidates selected uniformly in the interval* $30 \div 350$ *(with alpha= 0.3). This plot averages 10000 experiments for each curve in the described scenario.*

On the other hand, the learners with higher number of candidates seem to converge to a better result overall despite collecting a considerably bigger amount of regret. This makes sense since **the more candidates we have to explore, the bigger is the probability to select a good candidate but also the longer it takes to discriminate between prices**. In fact, if we only explore a few arms, we will

most probably end up with sub-optimal candidates, however in a very short time. As an example, if we consider the learner with 5 arms, we can see that when its exploration phase ends, almost immediately it surpasses the learners with 10 and 17 arms but then it settles around a quite small reward value. Indeed the optimal value of this learner is 64.32 €, which is way smaller than the optimal one for the other two learners.

If we now take a look at the learner with 4 candidates, it seems that its curve behaves in an unexpected way. In fact, during the exploration phase, the learner collects very small regret but in the end still manages to converge to a really high expected reward.
This happens because of mainly two reasons which can be easily explicable by looking at the values of the four candidates (18.76€, **67.54€**, 54.60€, 30.76€). The first reason is that one of the arms (the optimal one) has a really high value that is almost equal to the best possible one. The second reason is that the other three values are, in fact, really far from the optimal. That's why the algorithm has no problem finding the best one among the four and, as a matter of fact, in this example always chooses the correct arm for the exploitation stage. This can also be seen in Figure 15 where the regret sharply becomes zero and is then overtaken by the 5-arm learner one.

From this brief analysis we can say that, most probably, a good choice for the company would be to select a configuration that collects a regret that is under a prefixed amount but that still yields good results for the long term. In fact, if we select a bad arm, the losses in the long time horizon might be worse than the ones collected during the exploration stage. Therefore it might be important to sacrifice some profits in favor of better future rewards. In this example, except for the lucky case of the 4-arms configuration, **the 10-arms learner could be a suitable compromise** between performance and losses.

Let's assume now to have a **fixed amount of time available for our analysis**, that in our case is the usual 18250 samples. If we plot the results of the different configurations of arms, we will notice a slight difference in behaviour with respect to the previous example.
This time, since we have the same number of exploration samples available for all the learners, we face a new problem as we increase the number of arms to explore. In fact, as we need to confront more candidates, we have increasingly less samples at hand for each hypothesis test. Therefore a configuration with a small number of arms (for example 4 and 5) estimates way better it's optimal value since it has a significant bigger amount of samples for each step, as we can see in Figure 16.

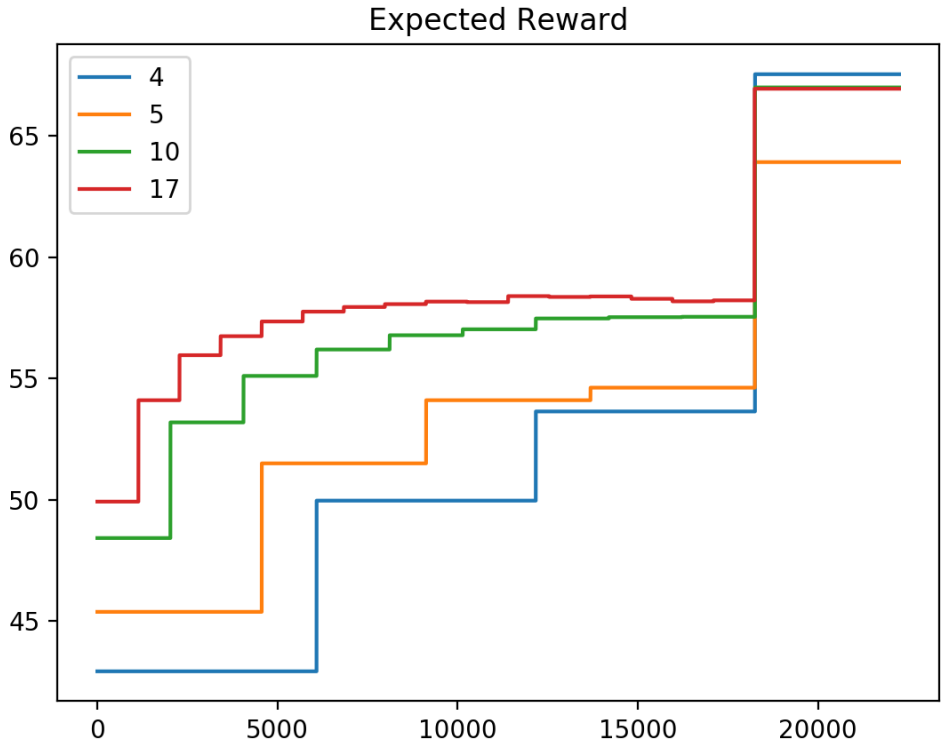

Figure 16: *Expected reward of the algorithm with different numbers of candidates in the fixed time horizon (with alpha= 0.3). This plot averages 10000 experiments for each curve in the described scenario.*

zContrarily, we have that **the configurations with more arms are increasingly less precise due to the lack of samples.**
Looking at the cumulative regret plot (Figure 17) it seems we have the inverse of what was happening previously. In reality the situation is similar except for the fact that the exploration phase is equally long in all the learner and thus the regret collected during exploration is lower as we increase the arms but the one during exploitation is best with less candidates.
Despite these differences, some considerations still hold from the previous example. Indeed, we still have that choosing a learner with a big number of candidates is more likely to end up finding an optimal price and vice versa. For this reason we think that the 10-arms learner is still a good choice overall.

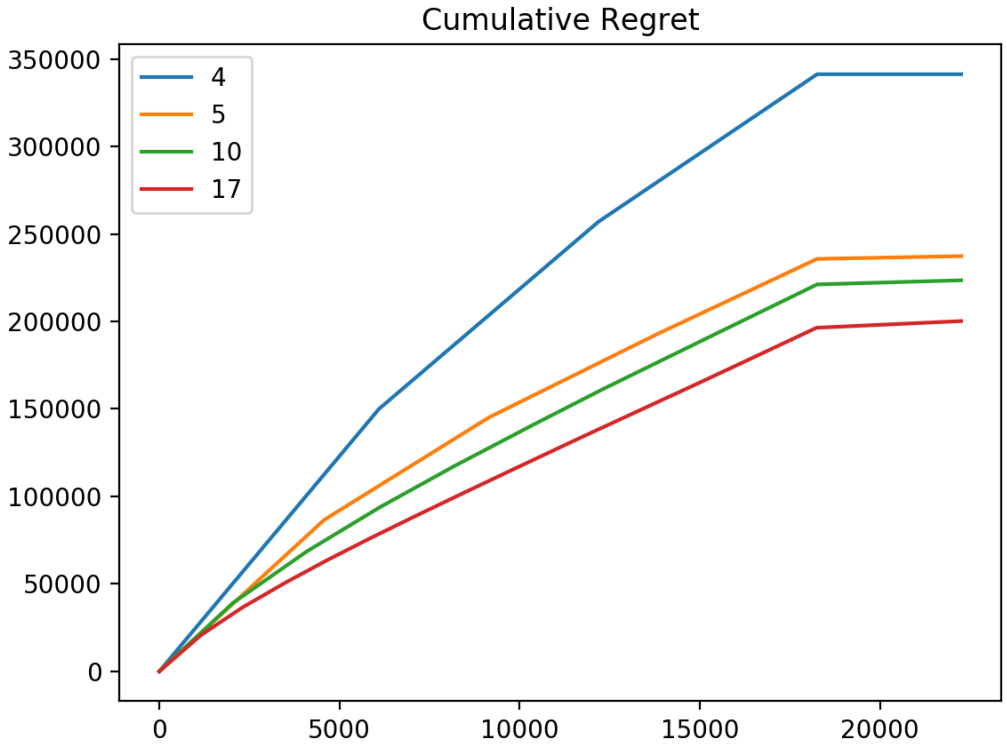

Figure 17: *Cumulative regret collected by the algorithm with different numbers of candidates in the fixed time horizon (with alpha= 0.3). This plot averages 10000 experiments for each curve in the described scenario.*

Non-Stationary Performance

We know from the theory that **the k-testing algorithm is not equipped with the tools needed to deal with non-stationary environments**. Indeed, we have seen that **once we discard a candidate** at any given time of the process, **we can never select it in the future**. This is a big problem if we know that we have a seasonality in our demand curve (like in our case).
For example, let's say we have a candidate that is optimal only during a specific phase but otherwise it's sub-optimal. **If our algorithm is executed between two or more phases, we may discard an optimal candidate** because we collected its samples during a sub-optimal stage.
One thing that can be done to partially solve this problem **is to run several k-testing experiments, one for each phase of interest**. This way we make

sure that each experiment is executed in a stationary environment and thus we try to find the optimal candidates for each demand phase.

In the following plots we show the performance of k-testing in the three different phases (low, medium, high) with the best parameters found so far (n°arms = 10, alpha = 0.3).

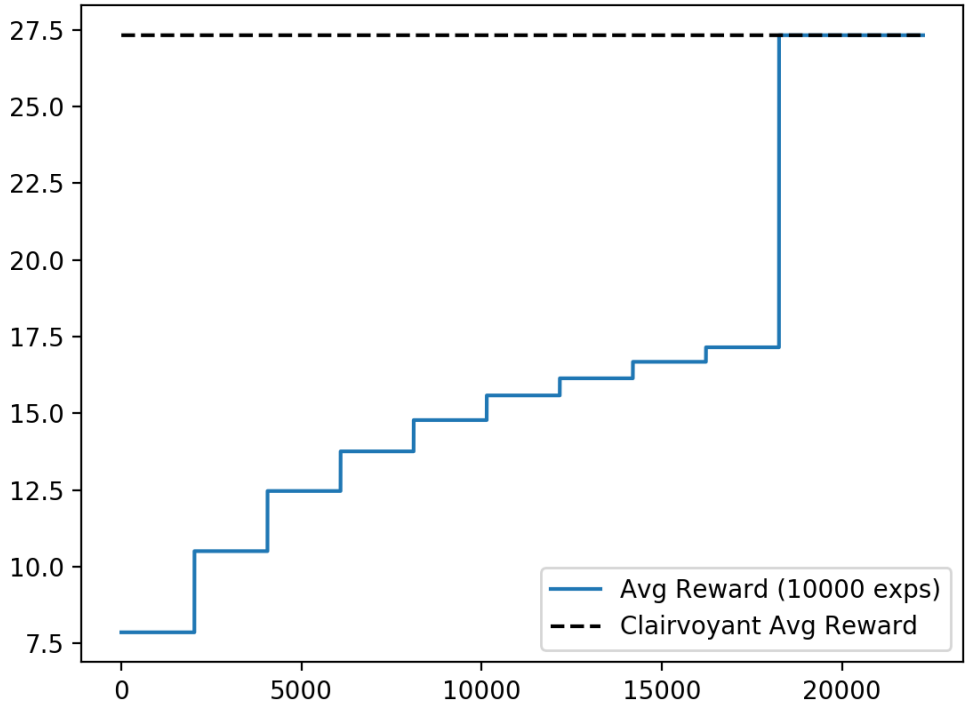

Figure 18: *Expected reward of the algorithm with 10 candidates, alpha = 0.3, phase of low interest in the fixed time horizon. This plot averages 10000 experiments for each curve in the described scenario.*

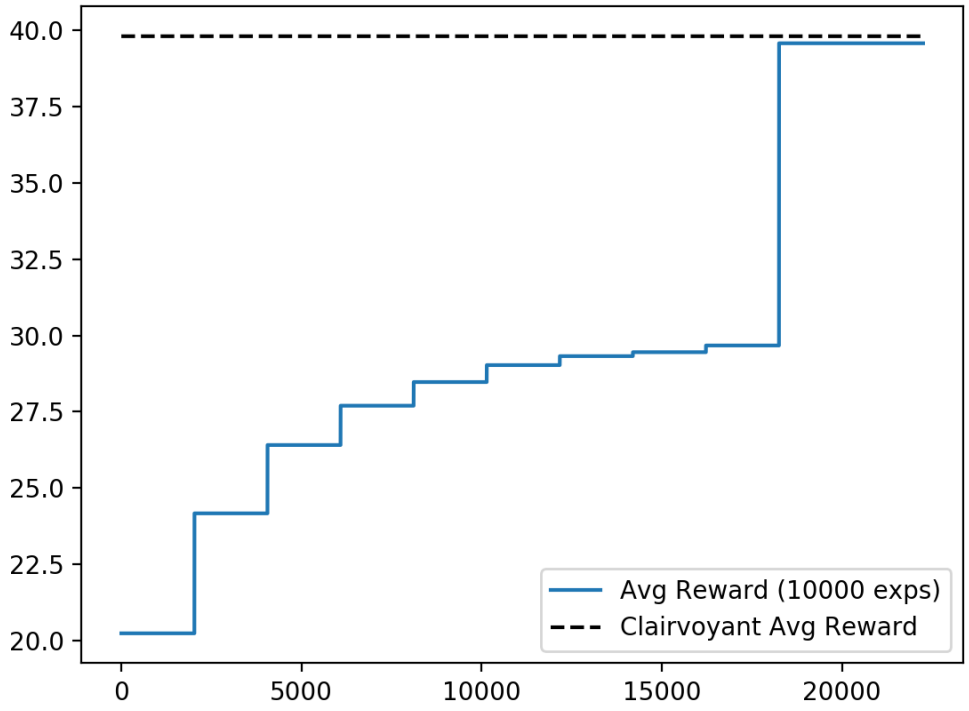

Figure 19: *Expected reward of the algorithm with 10 candidates, alpha = 0.3, phase of medium interest in the fixed time horizon. This plot averages 10000 experiments for each curve in the described scenario.*

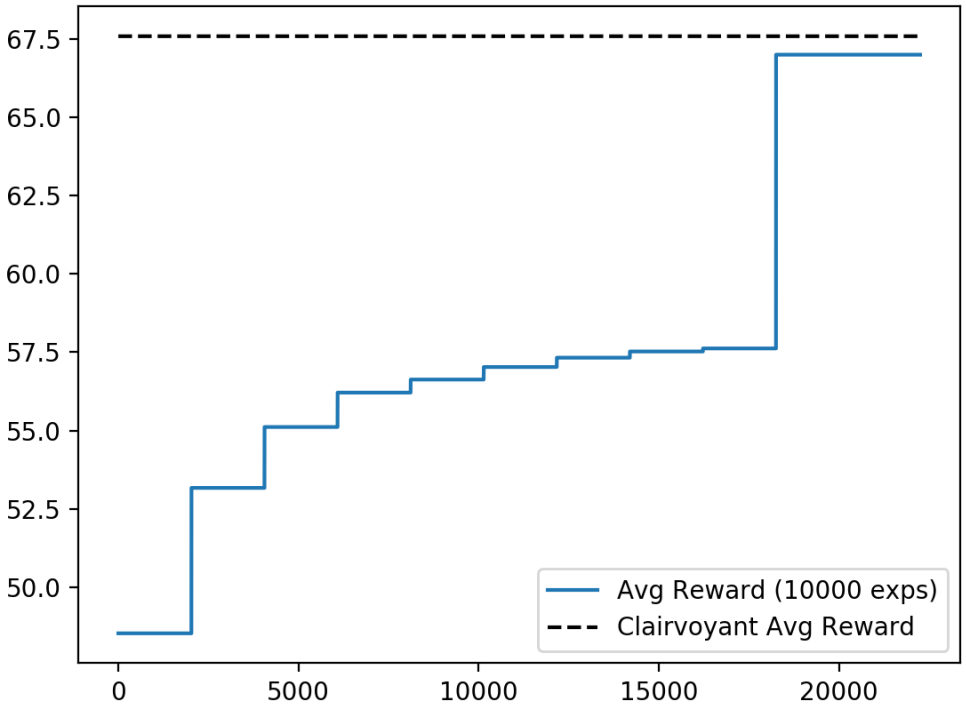

Figure 20: *Expected reward of the algorithm with 10 candidates, alpha = 0.3, phase of high interest in the fixed time horizon. This plot averages 10000 experiments for each curve in the described scenario.*

5.3 UCB1 Algorithm

In this section we will analyze the performances of both regular UCB1 and UCB1 with sliding window, in different scenarios.
For the time being, let's suppose that the demand curve, displayed in Figure 2, does not change during time. In other words, let's consider a **stationary environment scenario.**
The pseudo-code of a generic regular UCB1 algorithm in case of profit maximization is here reported:

1. Play every arm $a \in A$ once;
2. At every time t play arm $\overline{a}$ such that:

$$\overline{a} \leftarrow \underset{a \in A}{\operatorname{argmax}} \left\{ Pr_a * \left(\overline{x}_a + B * \sqrt{\frac{2 * \log t}{N_{a,(t-1)}}} \right) \right\}$$

Where:

- $\overline{x}_a$ is the empirical mean of the arm a;
- Pr_a is the profit of the arm a, which can be easily computed by subtracting from the candidate relative to the arm a the production cost (in our case it's fixed and equal to 10€).
- B is a scale factor for the bound of every arm. It plays an important role in terms of convergence speed of the algorithm and robustness in case of non-stationary environment.
- $N_{a,(t-1)}$ represents the number of collected samples for the arm a up to time $(t-1)$.

As mentioned in Section 4, the number of arms, given the number of interactions (in our case: $T = 18250$), should be at most 17 in order to obtain an acceptable result in terms of performances in that pre-established time horizon.

How The Scale Factor B Affects The Performances

As just mentioned, the scale factor B for the bound plays an important role in terms of performances. Here we report, as an example, the plot that shows how the behaviour of a regular UCB1 learner with 17 arms changes depending on the values of B. As can be noticed by looking at Figure 21, the smaller is the number B, the

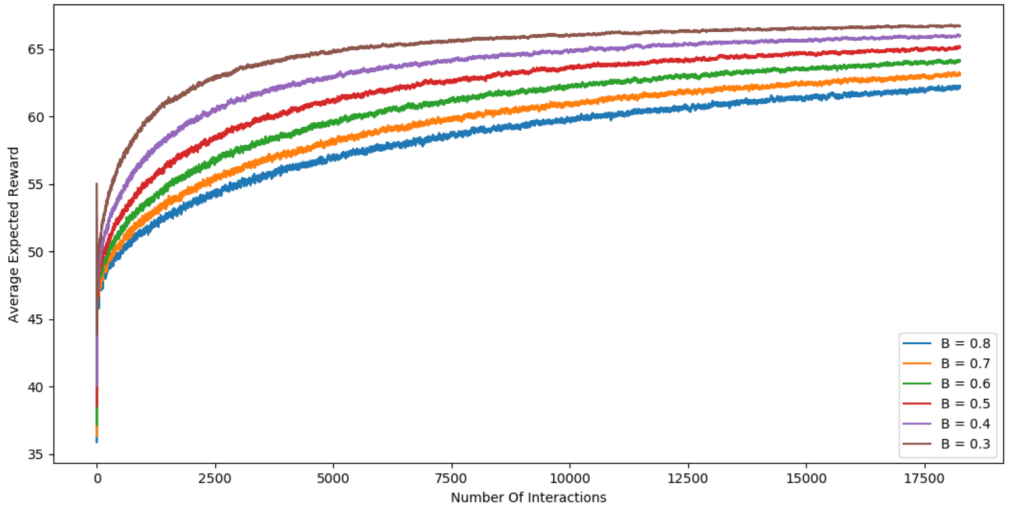

Figure 21: *Instantaneous expected rewards given by an UCB1 learner with 17 arms distributed uniformly in the interval* $30 \div 350$*, w.r.t. different values of the bound scale factor* B*. This picture has been made by averaging overall 10000 experiments in the described scenario. On the* X *axis we find the number of performed interactions with the environment, on the* Y *axis the average instantaneous expected reward of this learner.*

faster is the algorithm (in average) to converge to its optimal value of the average expected reward (that is represented by the marginal profit $MPr_{a'} = Pr_{a'} * \overline{X}_{a'}$, where a' is the arm such that: $a' \leftarrow \underset{a \in A}{\text{argmax}}\{Pr_a * \overline{X}_a\}$. Please, note that $\overline{X}_a$ is the theoretical expected value of the Bernoulli relative to arm a, while $\overline{x}_a$, as reported in the description of the UCB1 algorithm, is the empirical mean updated step by step by the learner with the performed interactions with the environment). However, the smaller is the value of B, the less robust will be the algorithm in case of non-stationary environment. In simple words, this value reflects how much effort we want to spend to explore the various arms with the algorithm. Is now clear that, in case of stationary environment, a small value of B allows us to rapidly identify the best arm by giving less importance to the exploration aspect; on the other hand, the exploration aspect becomes fundamental in case of non-stationary environment, therefore small values of B in this kind of scenarios lead to bad performances. **From this point on, we're using as a value of the factor** B **the one which gives us the fastest convergence, which is equal to** 0.3.

Introductory Example

As first example we chose to use 17 arms to have an idea of how the UCB1 algorithm behaves by working with this large number of arms. Figure 22 shows the instanta-

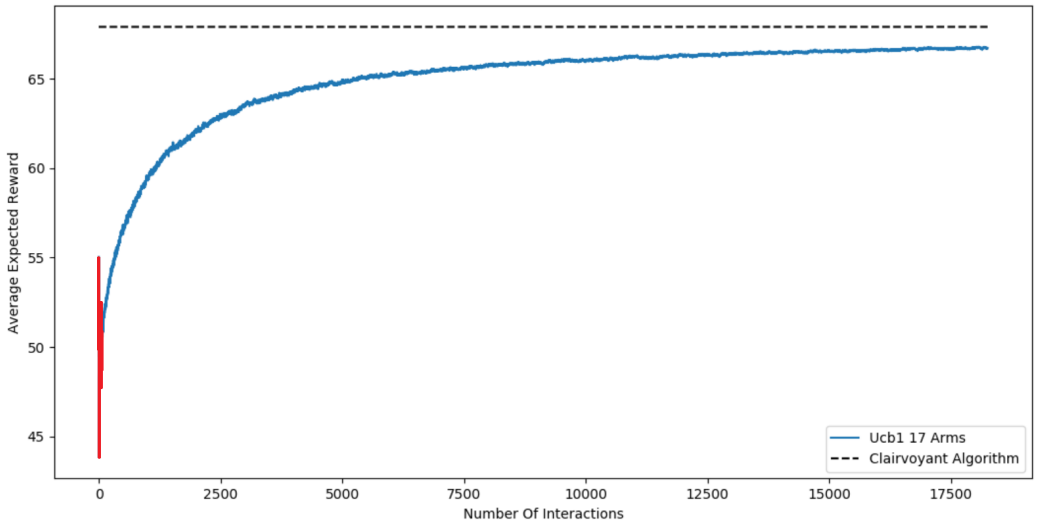

Figure 22: *Instantaneous expected reward given by an UCB1 learner with 17 arms distributed uniformly in the interval* $30 \div 350$. *This picture has been made by averaging overall 10000 experiments in the described scenario. On the X axis we find the number of performed interactions with the environment, on the Y axis the average instantaneous expected reward of this learner.*

neous expected reward (in red and blue) given by an UCB1 learner working with such a number of arms and the instantaneous expected reward (dashed black line) given by the clairvoyant algorithm.
The clairvoyant algorithm identifies the optimal value for the average expected reward of this learner that is, in this scenario, equal to $Pr_7 * \overline{X}_7 = 67.89$€. It corresponds to the 7-th arm (the relative candidate is the price: 150 €), the optimal one among those reported in Table 5.
Looking at Figure 22 it can be easily noticed that, at the beginning of the learning process, the UCB1 learner has no knowledge about which arm to pull. This is reflected by the portion of the function highlighted in red, that is characterized by those typical oscillations: they indicate the "uncertainty" of the learner in this early phase of the learning process. As the time flows, the learner acquires more knowledge about the goodness of every arm and the average expected reward increases towards an asymptotic convergence to its optimal value (portion of function highlighted in blue).

At the end of the process we can easily see that the average expected reward is way below the optimal one, this is typically due to the fact we chose a number of arms which is not so small (considering the number of interactions), it is a value suggested by the theory in order to achieve acceptable results, by taking in account the trade-off behind the choice of the number of arms (which will be discussed later). The phenomenon of non-achieving the best marginal profit reward (within the number of interactions) is also reflected by the plot of the cumulative average regret relative to this UCB1 learner. As we can see in Figure 23 the final phases of the

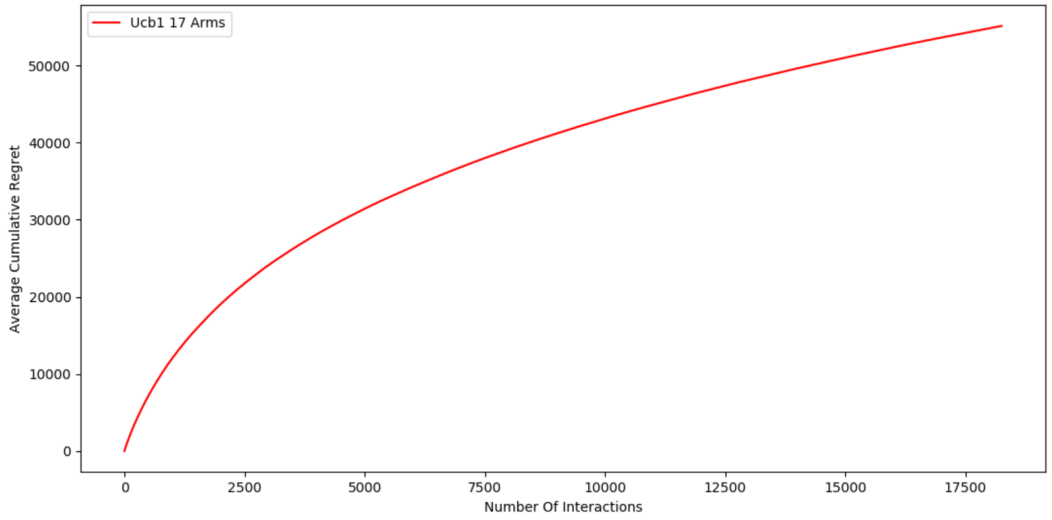

Figure 23: *Cumulative regret given by an UCB1 learner with 17 arms distributed uniformly in the interval* $30 \div 350$. *This picture has been made by averaging overall 10000 experiments in the described scenario. On the X axis we find the number of performed interactions with the environment, on the Y axis the average cumulative regret of this learner.*

process are not characterized by a convergence of the cumulative regret, this means that, in average, $T = 18250$ is not a large enough number of interactions to see the learner constantly pulling the optimal arm towards the end of the time horizon. In conclusion, even though 17, that is, the upper-bound on the number of arms, is acceptable according to the theory, we do not suggest to use numbers too close to the theoretical upper-bound.

Playing With Different Numbers Of Arms

As mentioned few lines above, the choice of the number of arms is one of the most critical parts in this kind of applications. Behind this choice there is a trade-off that

strictly depends on the time horizon over which we, as a company, are willing to optimize:

- Choosing a larger number of arms implies a slower convergence to the optimal one (the learner has to explore for longer time to identify the best arm). On the other hand, it gives us more chances to select the best price as a candidate among all the possible prices;
- Choosing a smaller number of arms implies a faster convergence to the optimal one, but it gives us less chances to select the best price as a candidate.

To better understand this concepts let's now consider two additional UCB1 learners, with 10 and 5 arms (we decided to uniformly distribute these new arms over the interval $30 \div 350$ assuming we have no knowledge about the best price). The values of the new candidates, as well as the corresponding Bernoulli's expected values and Marginal profits are reported in Table 3 and Table 4 (in each table the bold row represents the optimal arm). Chosen the arms, we can now make a comparison of the performances of these different learners. Figure 24 shows how the different

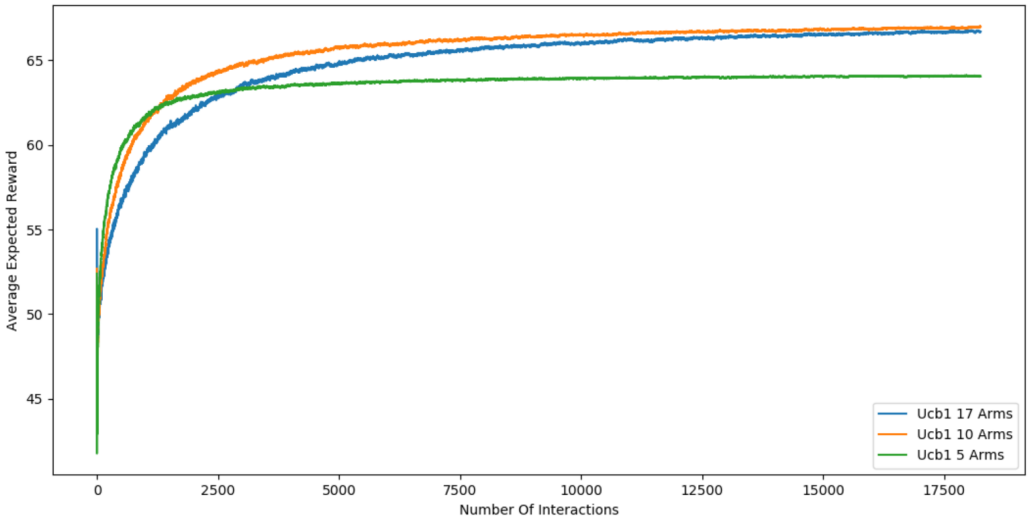

Figure 24: *Comparison between the average expected rewards of learners with different number of arms. This picture has been made by averaging overall 10000 experiments for each learner. On the X axis we find the number of performed interactions with the environment, on the Y axis the average instantaneous expected reward of these learners.*

average exp. reward curves evolve in time. As expected by the theory, the learner with 5 arms is the fastest to converge to its optimal value of the average expected reward that is, as reported in Table 3, equal to $Pr_3 * \overline{X}_3 = 64.32€$ (in average, it requires around 7000 interactions to have the learner constantly pulling the best

arm). However, the best arm found by this learner provides the worst marginal profit if compared with the others, given by the 17 and 10-arms learners (which are respectively equal to $Pr_7 * \overline{X}_7 = 67.89$ € and $Pr_4 * \overline{X}_4 = 67.59$€). This is typically due to the fact that, with such a small number of arms, we're using a too coarse grained selection for the candidates, therefore prices around 150 € (which is the best price identified by using 17 candidates) are not chosen. The 17 arms-learner, on the contrary, is the slowest to converge to its optimal arm, but the marginal profit that the latter provides is the best among the 3 considered in this scenario. We can also plot the regrets of these learners to have an idea of their values and how they evolve with the interactions. Figure 25 shows the comparison of the cumulative regrets between the 3 learners. As expected, the larger is the number of arms, the bigger is going to be the accumulated regret. Furthermore, the green curve hints to a convergence towards a value slightly above 10000. This confirms what we saw in Figure 24, where the green curve has almost reached, in average and within the end of the time horizon, its optimal reward. Seemingly, the other two learners don't show a convergence in their regrets.

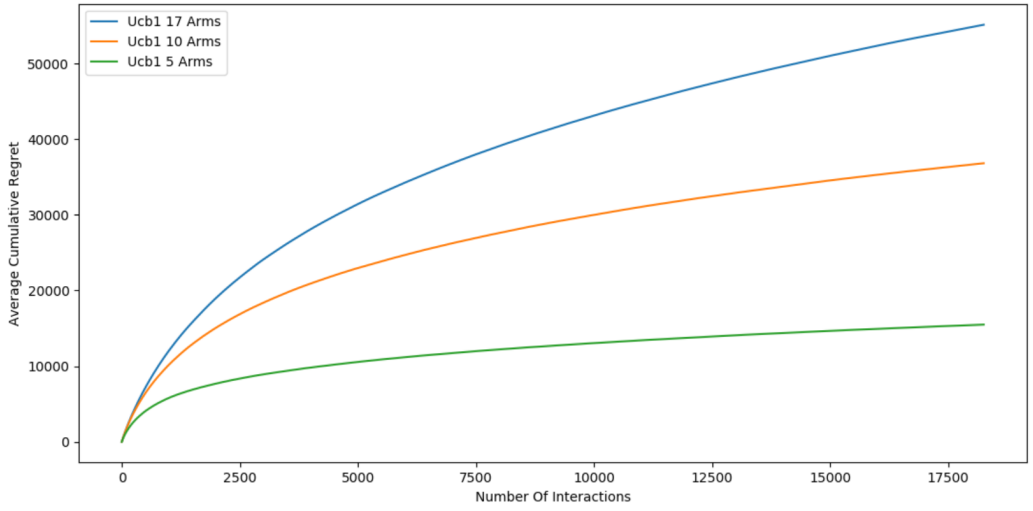

Figure 25: *Comparison between the cumulative regrets of learners with different number of arms. This picture has been made by averaging overall 10000 experiments for each learner. On the X axis we find the number of performed interactions with the environment, on the Y axis the average cumulative regret of these learners.*

How To Practically Choose A Proper Number Of Arms

How can then we, as a company, choose among learners with different number of arms? The choice mostly depends on how long the time interval over which we

want to optimize is. To better understand how we can select a learner rather than another let's consider the function $G_{L1,L2}(k)$, described in Section 5.1. We are now considering L1 as the UCB1 learner with 17 arms and L2 as the UCB1 learner with 5 arms (whose data are reported respectively in Table 5 and Table 3), we can plot the function $G_{L1,L2}(k)$:

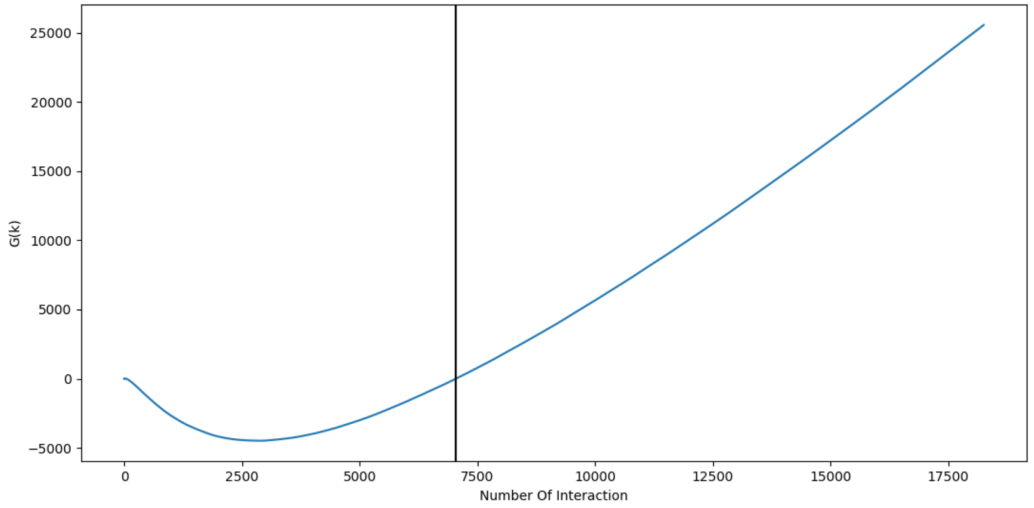

Figure 26: *Function G for learners L1 and L2, respectively equal to 17-arms-learner and 5-arms-learner. This picture has been made by averaging overall 10000 experiments for each learner. On the X axis we find the number of performed interactions with the environment (i.e the value "k" of the $G_{L1,L2}(k)$ function), on the Y axis the latter's values.*

Figure 26 shows the plot of the function mentioned above. As can be noticed, the curve can be divided in two parts:

1. In the first part, from interaction $k_1 = 0$ to interaction $k_2 \approx 2800$, the function $G_{L1,L2}(k)$ is decreasing. In other words, in these early phases of the learning process, we are loosing money by optimizing with the learner with 17 arms rather than the one with 5 arms. This is typically due to the fact that, in average, the learner with 5 arms rapidly converges to its optimal value for the average expected reward, while the learner with 17 arms is slower. For this reason it turns out that, in the beginning of the learning process, we accumulate more reward with the learner with 5 arms with respect to the learner with 17 arms. This fact makes then the function $G_{L1,L2}(k)$ decrease during time.

2. The interaction $\overline{k} \approx 2800$ identifies the point at which what we are gaining with the 17-arms-learner is equal to what we are gaining with the 5-arms-learner

(that is, the point in which the first derivative is null). From this point on we can recognize the second part of the curve, in which the function $G_{L1,L2}(k)$ has positive derivative, thus is increasing in time. The meaning of this fact is that, starting from there, the 17-arms-learner starts recovering all the deficit of money accumulated with respect the 5-arms-learners in the early phases of the learning process.

It's important to remind that, as reported in the corresponding tables, the learner with 17-arms provides the greatest optimal marginal profit. Given that, with enough time, this learner will start accumulating more reward than the 5-arms-learner, thus at some point the two accumulated rewards will be equal (the target interaction in the example above is marked by the vertical black line). We call that point (i.e. the target interaction) **Break-even point**, that is the point at which we, as a company, can start seeing an actual difference for having optimized with a larger number of arms, and it is identified as the value $\overline{k}$ such that: $\overline{k} \neq 0 \wedge G_{L1,L2}(\overline{k}) = 0$. In simple words, we can define the break-even point as the interaction at which the 17-arms-learner has recovered all the deficit of money accumulated with respect to its competitor with 5 arms. From that point on, the 17-arms-learner can make the difference due to its larger number of arms by accumulating more reward.
In this example, the break-even point takes place around 7050 interactions that are, under our hypothesis to receive 50 visits per day, approximately 4 months and two weeks. Are we, as a company, willing to wait this time to see an evidence for having chosen 17 arms rather than 4? If we don't, there is no reason to choose such a large number of arms, we can basically choose 4. What this example would like to show (and this is what is done in practice) is the correlation between the choice of the number of arms and the time span that we choose to optimize with our UCB1 algorithm. We can't select a large number of arms and a too short time horizon (and vice-versa): we have to find an affordable trade-off between time horizon and number of arms which, in practice, is not always a trivial task.

Let's now consider a more interesting example to be analyzed with the $G_{L1,L2}(k)$ function. As concluded before, selecting a number of arms which is too close to the value suggested by the theory is not a good choice. The plot of the corresponding curve is reported in Figure 27, in which the 10-arms-learner is selected as L1 and the 5-arms-learners is selected as L2. As in the previous examples, it is characterized by an initial decreasing behaviour for the same reasons. Two facts need to be underlined:

1. The break-even point (highlighted by the vertical black line), on the contrary to what is shown in Figure 26, takes place around 2760 interactions, which correspond (under our assumptions) approximately to 2 months.

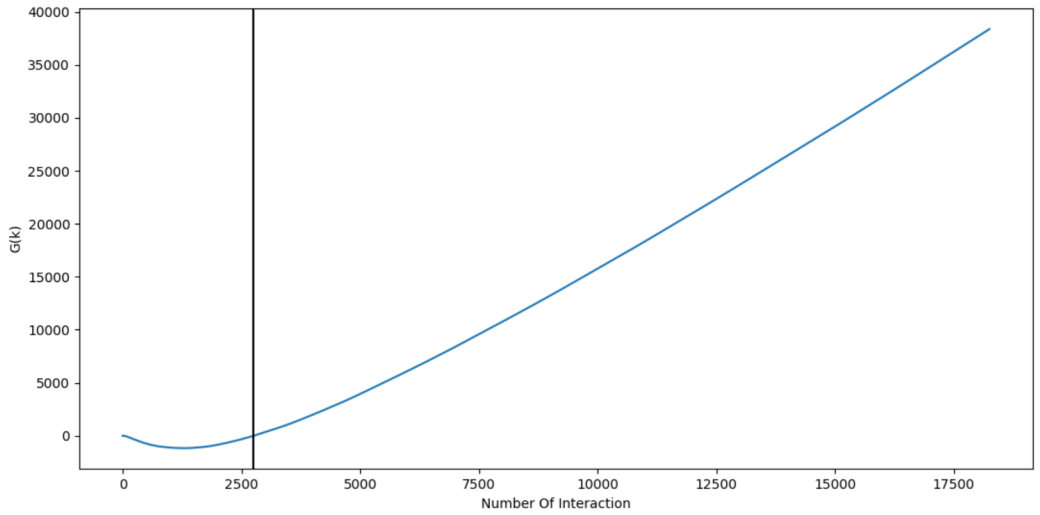

Figure 27: *Function G for learners L1 and L2, respectively equal to 10-arms-learner and 5-arms-learner. This picture has been made by averaging overall 1000 experiments for each learner. On the X axis we find the number of performed interactions with the environment (i.e the value "k" of the $G_{L1,L2}(k)$ function), on the Y axis the $G_{L1,L2}(k)$ function's values.*

2. The 10-arms-learner's average expected reward increases faster towards its optimal value if compared with its counterpart, that is relative to the 17-arms-learner. This fact causes a reduction of the interactions needed to see a break-even point (2760 interactions vs 7050 interactions), if both the 17 and 10 arms learner are compared with the same competitor (i.e. in this case the learner with 5 arms).

Since we consider 2 months a reasonable time horizon for our company, a 10-arms-learner is preferred to its counterpart with 5 arms.

To complete the analysis with the $G_{L1,L2}(k)$ function we're now going to compare the 17-arms-learner, selected as L1, and the 10-arms-learner, selected as L2.

Figure 28 shows the curve of the function $G_{L1,L2}(k)$. Few considerations:

- As expected, the early phases of the learning process are associated with a decreasing behaviour of the $G_{L1,L2}(k)$ function. As already discussed, this is due to the fact that the learner with 10 arms converges faster to the optimal marginal profit than its competitor with 17 arms.
- As can be easily noticed $G_{L1,L2}(k) < 0,\ \forall k \in (0, 18250]$, therefore the break-even point does not take place within 18250 interactions, which are, under the hypothesis to receive 50 visits per day, $\approx$ **1 year**.

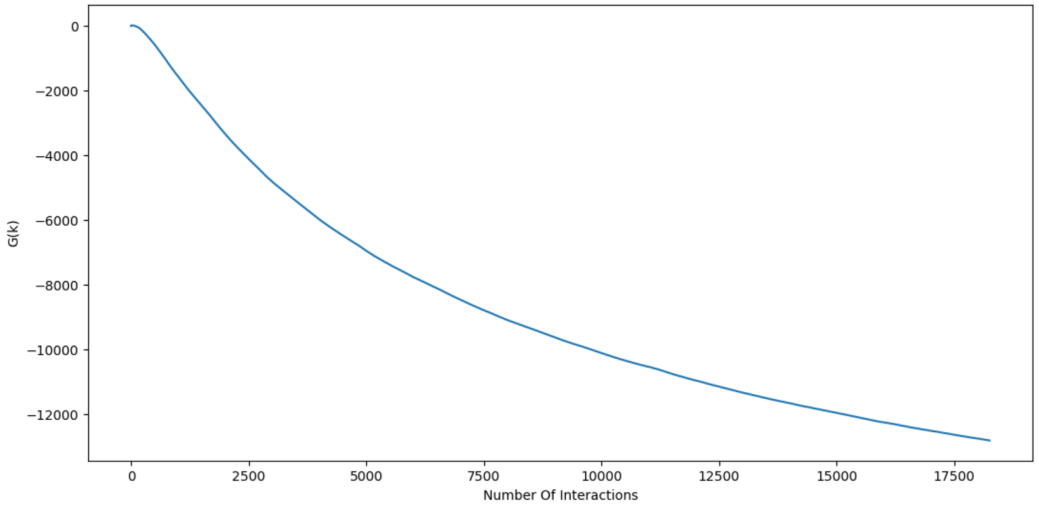

Figure 28: *Function G for learners L1 and L2, respectively equal to 17-arms-learner and 10-arms-learner. This picture has been made by averaging overall 1000 experiments for each learner. On the X axis we find the number of performed interactions with the environment (i.e the value "k" of the $G_{L1,L2}(k)$ function), on the Y axis the latter's values.*

- The reason why the break-even point requires so much time to take place is due to the large difference in the number of arms the learners have and the really small difference between the optimal marginal profits provided by the two learners. As the relative tables report, the 17-arms-learner's optimal marginal profit is equal to 67.89 €, while the 10-arms-learner's one is equal to 67.59 €. What should be derived from this point is that, if we would choose 17 arms, we are using 7 more arms to increase our best reward of $67.89€ - 67.59€ = 0.30€$. We can basically conclude that 7 more arms, to obtain such a small gain in terms of optimal marginal profit, are definitely not worth it.

For these reasons we can discard the hypothesis to choose a learner with 17 arms, in favor of its counterpart with 10 arms.

What should have emerged from this analysis is the problem to find a reasonable trade-off (according to our needs) between number of arms and the length of the time horizon over which we optimize. All the $G_{L1,L2}(k)$ functions discussed so far are a good tools to determine whether to use a number of arms rather than another. Unfortunately, they are not known a-priori and companies cannot exploit them to achieve the same results we argued above. We know only that the smaller is the number of arms, the faster will be the convergence to the optimal arm, but gives us less chances to select the best price (and vice-versa). However, there are some cases in which, even selecting a small number of arms, one of the candidates may turn

out to be close enough (or even equal) to the optimal price. Here as an example, we consider a learner with 4 arms, whose candidates are uniformly distributed over the interval $30 \div 350$. The relative data are reported in Table 2. We can immediately notice that, by comparing the optimal arm of this learner with the optimal arm of the best learner we found in this section (i.e. the 10-arms-learner, please check the relative Table 4), the marginal profits of the two best arms are really close to each other (as well as the relative candidates, 138€ and 137€) , in particular we find that the difference is equal to : $67.59€ - 67.54€ = 0.04€$. Therefore, we can conclude that the 4-arms-learner guarantees us the fastest convergence to its optimal arm (in average), despite a negligible loss in terms of total gain. Furthermore, playing with 7 more arms to improve our gain of 0.04€, as already discussed, is definitely not worth it. In conclusion, what we have discussed so far has led us to state that the 10-arms-learner provides the best service, according to our needs, for our company. However, if we would have considered in our analysis also the 4-arms-learner, it would have been the winner. To close this first subsection, relative to a stationary environment scenario, we show the performances of the learner just introduced.

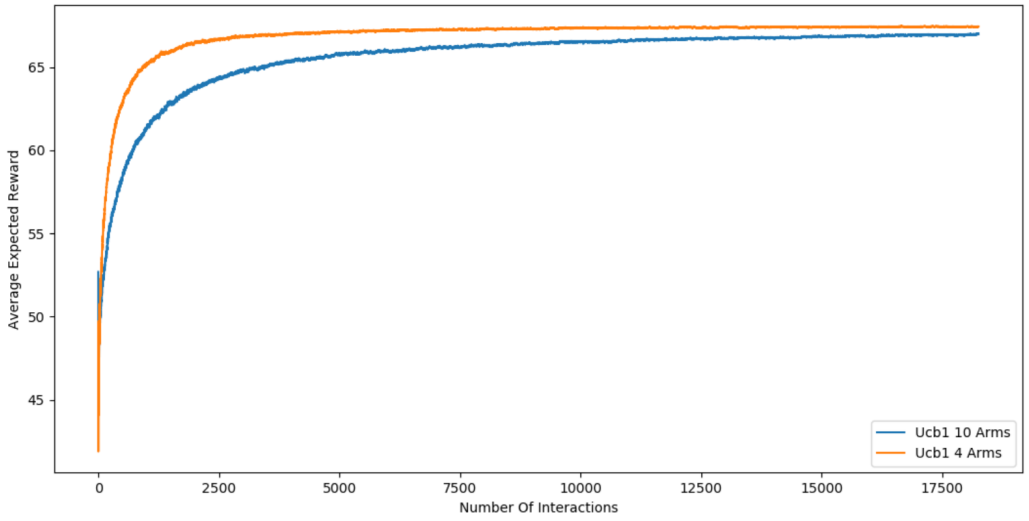

Figure 29: *Comparison between the average expected rewards of the 4-arms and 10-arms learners . This picture has been made by averaging overall 10000 experiments for each learner. On the X axis we find the number of performed interactions with the environment, on the Y axis the average expected rewards of these learners.*

Non-Stationary Environment Problem

What we have discussed so far relies on the fact that the demand curve does not change during time. In other words, the probability that a user, visiting our page, buys or not the product, given the price, is always the same. Unfortunately in real world this never happens, the behaviour of the customers frequently changes during time for various reasons (seasons, competitors...). To perform our analysis in this new scenario we will use the 3 different aggregate demand curves (each of them representing a specific phase of the changing environment: 'low', 'mid' and 'high'), which are displayed in Figure 6. For the sake of simplicity, being 3 the different demand curves, we are going to divide our number of interactions (i.e. 18250) in 3 parts, in each a different demand curve (i.e. a new set of Bernoulli expected values for the arms) is considered.
In this section we're going to compare the performances of the 17, 10 and 4 arms learners. By optimizing with these learners, the optimal arms in the different phases are reported in Table 8, Table 7 and Table 6. The regular UCB1 algorithm, in

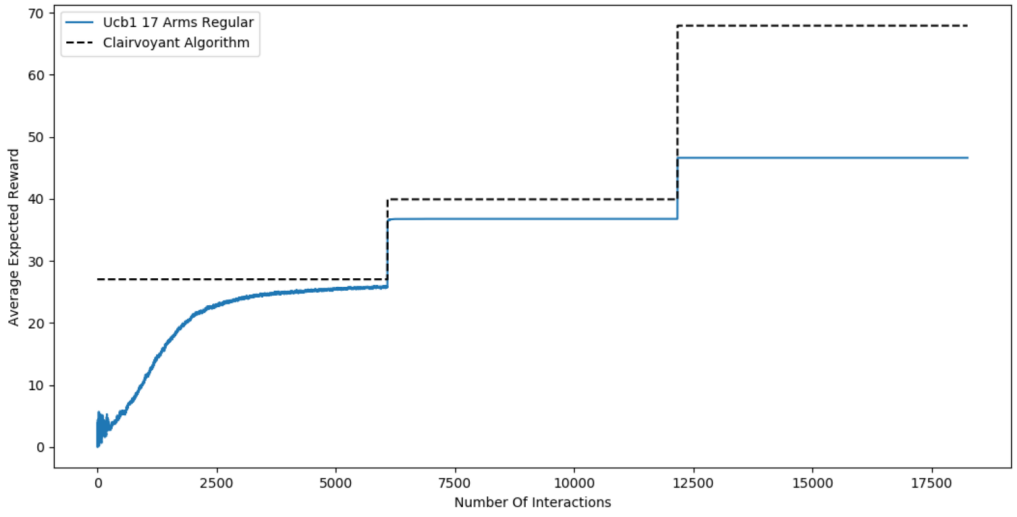

Figure 30: *Performance of a regular algorithm, playing with 17 arms, in a non-stationary environment. This picture has been made by averaging overall 10000 experiments. On the X axis we find the number of interactions performed with the environment, on the Y axis the average expected reward of this learner.*

this new kind of environment, does not work very well in all the possible cases. To better understand this point let's consider the following plot: Figure 30 shows how the average expected rewards of a regular UCB1 algorithms with 17 arms change during time, in a non-stationary environment. A few considerations:

- The clairvoyant algorithm (represented by the dashed black line in the picture) helps us to identify the three different phases, each of them characterized by a different optimal value for the average expected reward. In particular, as can be easily noticed, the phases follow the order 'low', 'med' and then 'high' interest.
- In the first phase, the average expected rewards grow normally, since the environment has not presented any change yet.
- The most relevant thing to be discussed is that, in the 'mid' and 'high' phases of this non-stationary environment, the algorithm does not realize that a change in the demand curve has taken place. In fact, by looking at Figure 30, it can be notice that in both 'mid' and 'high' phases of the environment, in average overall 10000 experiments, the learner always constantly pull an arm which is sub-optimal (this can be concluded by observing a perfectly flat behaviour of the average expected reward curve).

The regular UCB1 algorithm has no way to figure out whether an abrupt change has occurred or not, as well as to face the environment changes. For these reasons we are now going to analyze the performances of the UCB1 algorithm with sliding window in these kind of scenarios.

Different Ways To Select The Length Of The Window

Likewise the choice of the number of arms to play with, we are now required to face another problem: the choice of the length of the window for our UCB1 algorithm. Practically, the more we know about the environment (how and when abrupt changes occur, the number of the phases...) the more precise will be the window's length we're going to use. We're now going to make a comparison between the performances achieved by optimizing with the different sliding windows, computed with the formulas reported at the end of Section 4. Here as the first example we show the results obtained by a learner which is playing with 17 arms and using two sliding windows, whose lengths are given by $SW_1 = \sqrt{T}$ and $SW_2 = N * \sqrt{T}$. Being $T = 18250$ and $N = 17$, $SW_1 = 135$ and $SW_2 = 2296$. Figure 31 shows how the average expected reward given by this learner, adopting the just mentioned SWs, is varying over time. Some comments:

- Blue Curve, the first thing to be noticed is that this curve can be divided in two parts:
  - The dense part represents the values around which this learner spends most of the time during the learning process. Being this portion of the

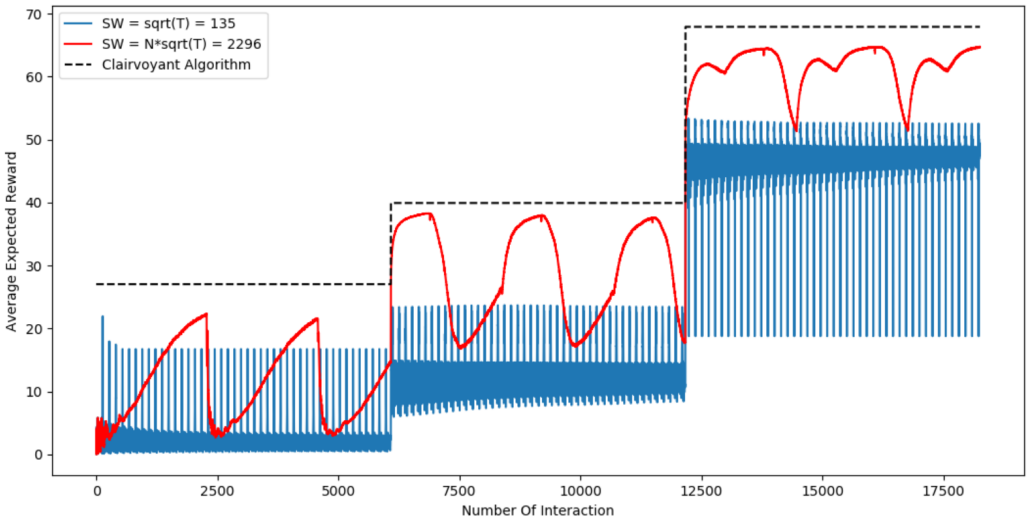

Figure 31: *Performance of a UCB1 algorithm playing with 17 arms and adopting two different sliding windows. This picture has been made by averaging overall 10000 experiments. On the X axis we find the number of interactions performed with the environment, on the Y axis the average expected reward of this learner.*

curve flat (in each phase), we can easily perceive the inability of this UCB1 learner to grow toward its optimal value of the average expected reward, whichever is the phase of the environment we are considering. Unfortunately, this fact leads to a huge accumulation of regret during time. The reason of this behaviour is given by the fact that 17 are a lot of arms to play with and 135 samples are too few to gather enough knowledge about every arm. In other words, the learner, playing with this large number of arms, has not enough time (it has only 135 interactions before starting to forget old samples) to acquire sufficient knowledge about the goodness of every arms, thus it is not able to improve its performances during time.

– The scattered part is composed by a set of peaks that take place whenever the learner has forgotten all the sample relative to a specific arm. When the UCB1 algorithm realizes an arm has no sample in the window, it is forced to pull that arm in order to always have at least one sample per arm in the window. The fact that the peaks, in the 'low' phase of the environment, are all above the dense portion of the curve should suggest us that the arms which are more frequent to remain without samples (in other words, the least pulled arms) are the more promising ones. This can be explained by considering the arms' Bernoulli expected values in

this first phase of the environment, which are very close to zero. This reason implies that the empirical means of every arm is very low and the arms with a bigger profit are preferred (unfortunately, such arms, which have a very high prices as a candidate, are not among the more promising ones).

  - Making a comparison with the regular UCB1 learner in the same scenario, we cannot see an actual improvement by adopting such a short window, the cumulative regret obtained by adopting SW_1 in even larger than the regular case (the corresponding regrets will be shown later).

- Red Curve:
  - The most relevant problem with the sliding window relative to the blue curve ($SW_1 = \sqrt{T}$) is due to the large number of arms. To overcome this problem we suppose that the sliding window length is directly proportional in the number of arms N and we assumed this dependency to be linear. The performances achieved by this new window are way better than what we just described for the other case, indeed the learner is able to approach the clairvoyant reward in every phase, accumulating less regret over time.
  - The length of this new sliding window allow the learner to collect sufficient information about every arm and then to improve its performances over time.
  - Improving the window length as described, we can see an actual improvement if we compare this new performances with the ones relative to the regular UCB1 learner in the same scenario.

As already underlined in Section 4 this two formulas used to compute the window length are exploitable also in real world scenarios since all the information required are known a-priori (the time horizon over which we optimize and the number of arms we're playing with). Let's now take a look to the other two formulas: the performances are expected to be way better than these just discussed, unfortunately they require information on the environment that are not known in advance. The learner we're considering is the usual 17-arms-learner, with $SW_3 = 2 * B * \frac{1}{F} * \sum_{f=1}^{F}(Rew_{max,f} - Rew_{min,f}) * \sqrt{\frac{T * \log T}{F-1}}$ and $SW_4 = \frac{T}{F}$.
Given $T = 18250$, $F = 3$, $B = 0.3$ and $\frac{1}{F} * \sum_{f=1}^{F}(Rew_{max,f} - Rew_{min,f}) = 37.814$, $SW_3 = 4474$ and $SW_4 = 6083$. Figure 32 shows the performances achieved by adopting these new lengths for the sliding window. As can be easily noticed, the performances in this scenario improve if compared with the ones obtained by adopting either SW_1 or SW_2. We don't have to forget that our objective is to find the

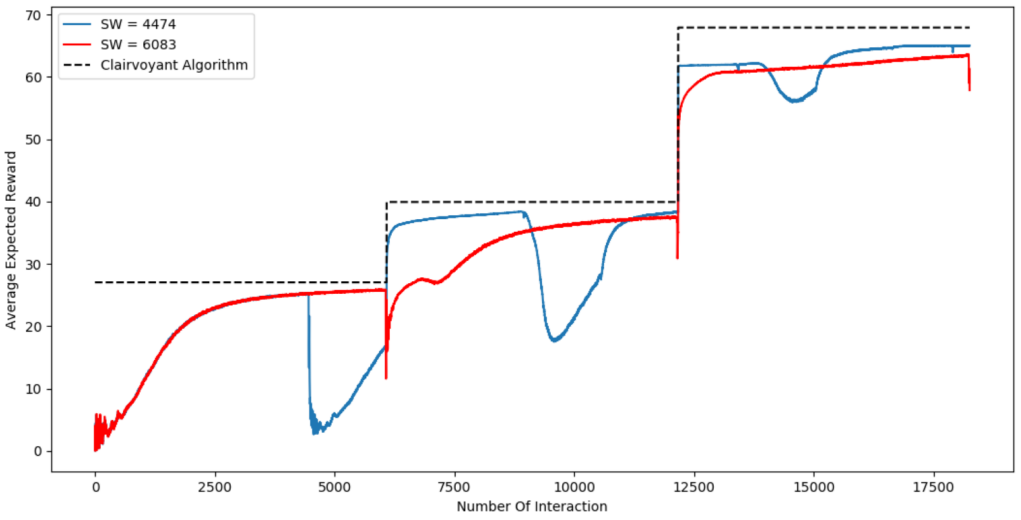

Figure 32: *Performance of an UCB1 algorithm playing with 17 arms and adopting two different sliding windows. This picture has been made by averaging overall 10000 experiments. On the X axis we find the number of interactions performed with the environment, on the Y axis the average expected reward of this learner.*

best sliding window that is, obviously, the one which minimizes the accumulated regret over the time horizon. To have more clear this point let's now take a look at how the different SWs length affect the accumulated regret. Unlike what might have been expected, adopting SW_1 does not led to the desired result. In fact, by looking at Figure 33, we can see how huge is the amount of regret accumulated by using this kind of window, even bigger than the case in which we do not use a sliding window. For this reason, in case we have no knowledge about the environment (which always happens in real world scenarios), we definitely discourage to use such a short window with so many arms. We suggest, indeed, to use SW_2, it does not require any information about the environment and it provides affordable performances during all the time horizon (Figure 32). As already mentioned, the more we know about the environment, the better will be the performances achieved by our learners. Windows like SW_3 and SW_4 require a lot of this information that in practice we don't know (especially SW_3), however they perform very well and are able to keep the accumulated regret at the lowest levels. In particular, SW_4 is the window which minimizes the regret among those analyzed so far. The reason of its success is due to the fact that its length is equal to the length of an environment's phase, thus the learner starts forgetting old samples (or in other words, the learner starts exploring other arms) in correspondence of an abrupt change of the environment, increasing the possibilities to recognize potential new best arms. It's easy to conclude that

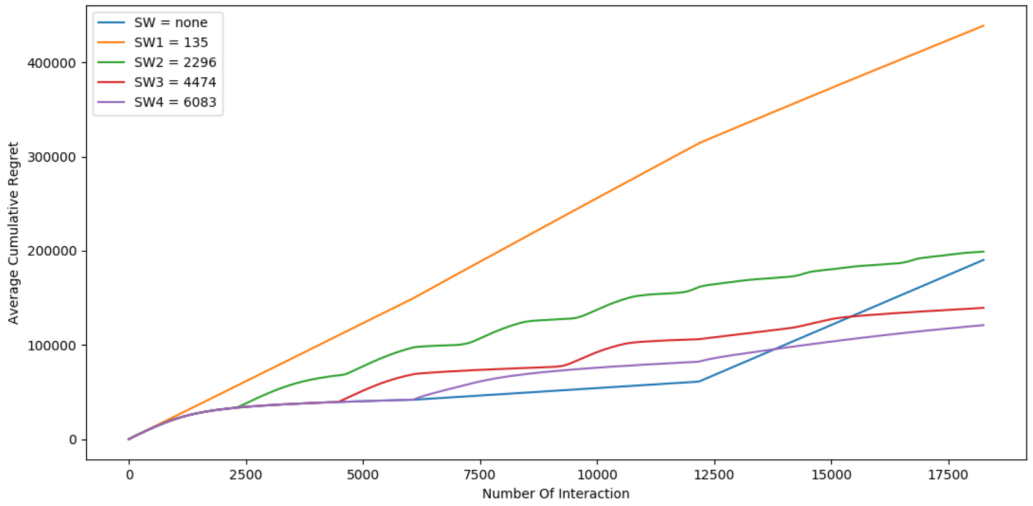

Figure 33: *Cumulative regret given by an UCB1 algorithm playing with 17 arms and adopting various kind of sliding windows. This picture has been made by averaging overall 10000 experiments. On the X axis we find the number of interactions performed with the environment, on the Y axis the average cumulative regret of this learner.*

this is the best we can do to, in average, minimize the regret. For shorter lengths (longer lengths), the learner starts exploring new arms before (after) the occurrence of a change in the environment, which can be translated in an accumulation of more regret than what is accumulated with SW_4.

Playing With Different Numbers Of Arms In Case Of Non-Stationary Environment

In this last subsection we're going to briefly discuss the changes introduced by playing with different numbers of arms. As we will see in a moment, the performances of the sliding windows are similar, while the convergence speed of the algorithm, as already analyzed, increases with a decreasing number of arms. The sliding window's lengths that will be used are the same as those used in the previous sub-section and reported in Section 4. Figure 34 shows how the average expected rewards and the corresponding regrets of a learner playing with 4 arms are varying over time. Likewise the 17-arms-learner performances, the most promising sliding windows lengths are given by SW_3 and SW_4. Unlike what we have discussed about the 4-arms-learner in case of stationary environment, in this new scenario the performances in the 'high' interest phase, without using a sliding window, are way worse than expected (in the left-hand side picture, the constantly pulled arm provides a very low average ex-

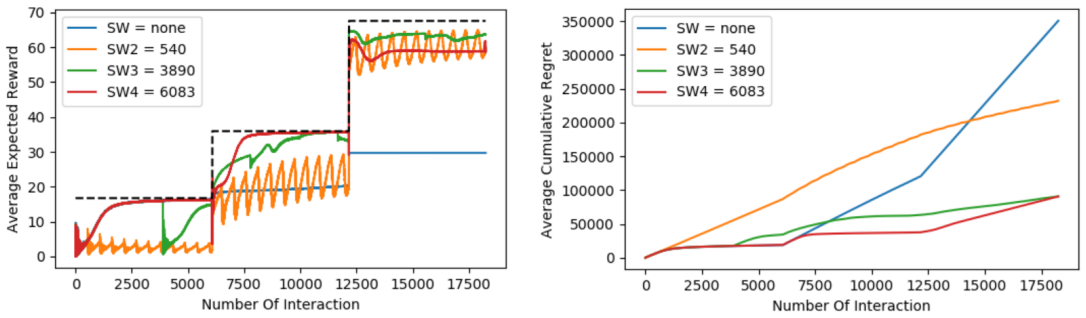

Figure 34: *Average expected reward and cumulative regret given by an UCB1 algorithm playing with 4 arms and adopting various kind of sliding windows. This picture has been made by averaging overall 10000 experiments. On the X axis we find the number of interactions performed with the environment, on the Y axis the average cumulative regret and the average expected reward of this learner.*

pected reward), which can be translated in a greater slope of the corresponding cumulative regret. This fact makes the adoption of a sliding window a forced choice if we want to have acceptable performances over the whole time horizon.

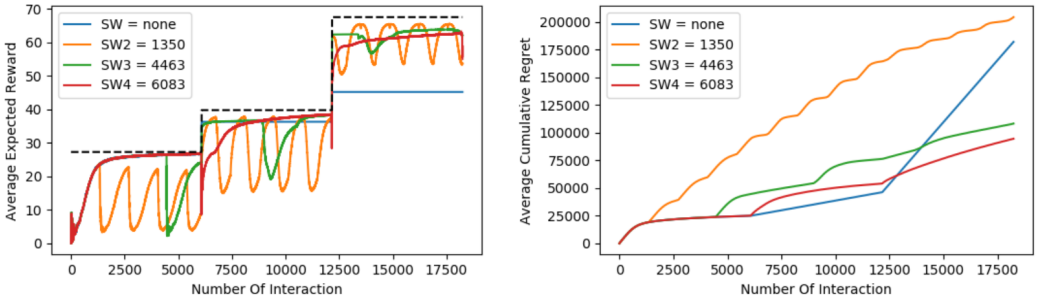

Figure 35: *Average expected reward and cumulative regret given by an UCB1 algorithm playing with 10 arms and adopting various kind of sliding windows. This picture has been made by averaging overall 10000 experiments. On the X axis we find the number of interactions performed with the environment, on the Y axis the average cumulative regret and the average expected reward of this learner.*

Figure 35 shows how the average expected rewards and the cumulative regrets of a learner playing with 10 arms are varying over time. The first thing that can be noticed is the cumulative regret given by $SW_2 = N * \sqrt{T}$. Surprisingly, it

turns out to be larger than the one given by the regular UCB1 algorithm. This behaviour is due to the fact that the regular algorithm does not under-perform too much in the 'low' and 'med' phases, while the SW_2-UCB1 algorithm, in the same phases, pays the price of exploring other arms (thanks to the window) and thus accumulates more regret. Obviously, in the last phase on this learning process, the SW_2-UCB1 learner's average expected reward is way above the sub-optimal regular UCB1 learner's reward. Therefore, with enough time, the cumulative regret of the latter will start to be larger than the former one, making the choice of SW_2 a good improvement for our company purposes. The performances achieved by SW_3 and SW_4 are, as expected, the best in this scenario as well.
At the end of this wide analysis on the non-stationary environment scenarios, we can conclude that all the examples we have seen so far make $SW_2 = N * \sqrt{T}$ a reliable window length choice for real world applications, achieving better performances compared with its counterpart $SW_1 = \sqrt{T}$.

5.4 Thompson Sampling Algorithm

In this section, we offer a brief explanation of the Thompson Sampling algorithm, followed by a presentation of the results we obtained by applying the Thompson Sampling algorithm in two different scenarios: in the first case, we will consider a stationary environment, that is, an environment in which the demand curve does not change over time; in the second case, we will consider a non-stationary environment, that is, an environment in which the demand curve changes over time. In both cases, different case studies will be analyzed and discussed. All results are obtained averaging over 10000 experiments.

The Algorithm

Thompson Sampling is an algorithm used to solve problems that can be modeled as Multi-Armed Bandit (MAB) problems. The arm to pull at each round is chosen based on a set of priors, that is, a set of (usually) Beta distributions holding the information gathered on each specific arm. One sample is drawn from each of these distributions and the arm corresponding to the highest sampled value is pulled. The prior distribution for that arm is updated according to the result of the pull, and the process is repeated. The idea is that prior distributions embed the information gathered on each arm, progressively holding more and more reliable information as the number of pulls for a specific arm grows. When drawing samples from the prior distributions, we are in fact considering the knowledge we collected so far, and then making a choice based on how convenient pulling a specific arm might be. This aspect is crucial to address the exploration-exploitation dilemma: poorly explored arms generally have a higher probability of sampling high values compared to more explored average-performing arms. Moreover, this probabilistic representation adds a layer of non-determinism. Unlike other algorithms, which choose the arm to pull at each round in a deterministic way, in Thompson Sampling even the worst-performing arm always has a small, almost insignificant chance of being pulled at each round. Contrary to being counterproductive, this might actually help the algorithm to recognize changes in the reward distributions in case of non-stationary environments.

We offer a formalization of the main steps of the algorithm below:

Init: $\forall a \; P(\mu_a = \theta_a) \sim B(1, 1)$

At every time t:

1. For every arm a: $\tilde{\theta_a} \leftarrow Sample(P(\mu_a = \theta_a))$

2. Play arm a such that: $a_t \leftarrow \underset{a \in A}{\text{argmax}} \{\tilde{\theta}_a\}$
3. Update the Beta distribution of arm a_t as: $(\alpha_{at}, \beta_{at}) \leftarrow (\alpha_{a_t}, \beta_{a_t}) + (x_{a_t,t}, 1 - x_{a_t,t})$

In our specific case, since we are dealing with a profit maximization scenario, in step 2 we play arm a such that $a_t \leftarrow \underset{a \in A}{\text{argmax}} \{\tilde{\theta}_a * Pr_a\}$, where Pr_a is the difference between the candidate price of arm a and the production cost, that is, the profit on one unit of product sold.

Stationary Environment

In a stationary environment, each pull provides useful information to recognize the best arm during the given time horizon. However, having to deal with a larger number of arms usually slows down the process, since there are more arms whose expected reward needs to be estimated. As a company, since we have no information

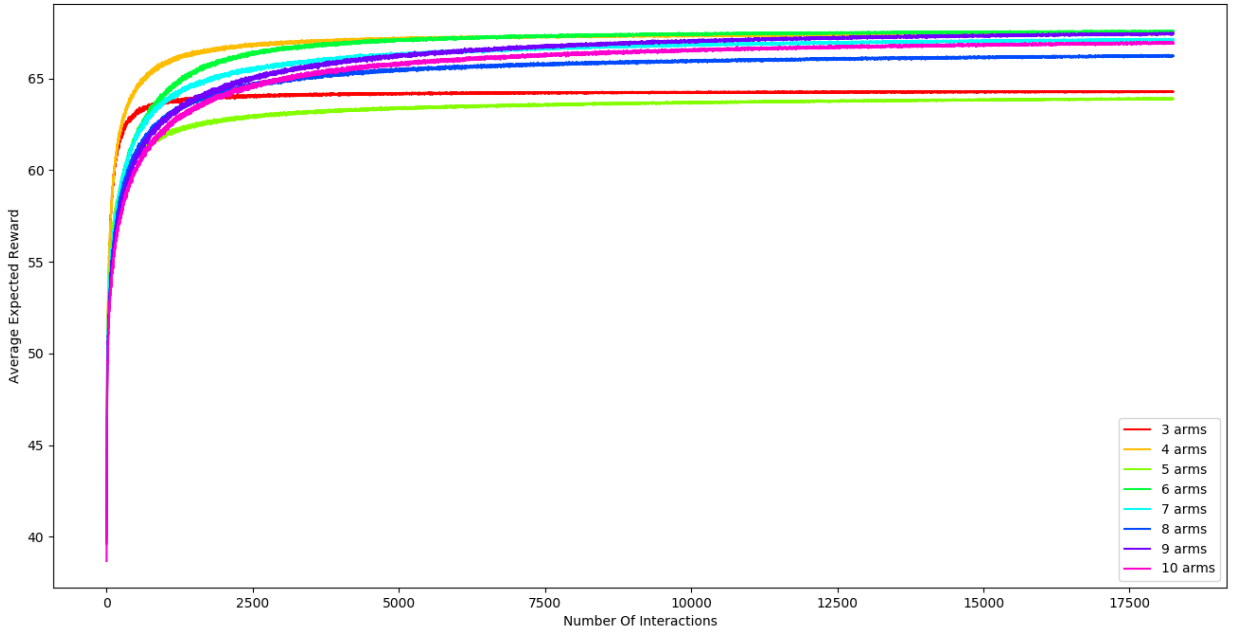

Figure 36: *Average expected reward convergence as the number of interactions increases for a number of arms ranging from 3 to 10.*

to rely on, the best choice is still to go for a number of arms that is close to the maximum we can handle, which is 17, as highlighted in Section 3. This makes it more probable that the optimal candidate of our batch is closer to the best candidate. However, we experimented with all numbers of arms going from 3 to 17 to see how the process would change.

In Fig. 36, we show the results obtained for a number of arms ranging from 3 to 10. Analyzing this first batch, we notice how convergence values for a little number of arms is generally lower. However, in this specific context, we notice how for a number of arms equal to 4, we are still be able to obtain one of the highest convergence values. This lucky event must not be extended to the general case. As one might expect, this is strongly related to the shape of the demand curve which, in real-world applications, we do not know exactly, as well as to the way in which candidates are selected. Choosing arms uniformly in a given range, as we did, generally produces a theoretical optimal profit that fluctuates around the best arm, converging to it as the number of arms goes to infinity.

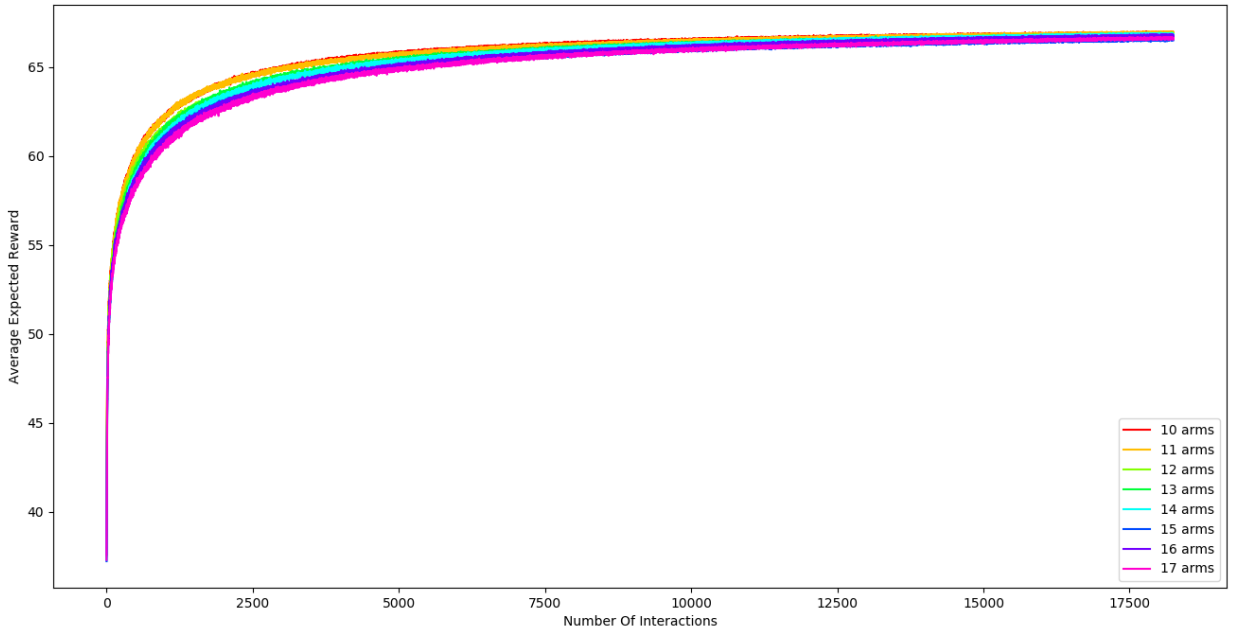

Figure 37: *Average expected reward convergence as the number of interactions increases for a number of arms ranging from 11 to 17.*

In Fig. 37, we show the results obtained for a number of arms ranging from 11 to

17. In this second batch, we notice how convergence has slowed down compared to the first batch, mainly due to the increased exploration required to manage a larger number of arms. The slow-down is directly proportional to the number of arms considered. Moreover, it is even more evident how we are converging to a similar optimum. Such a difference, in real-world scenarios, might be considered negligible.

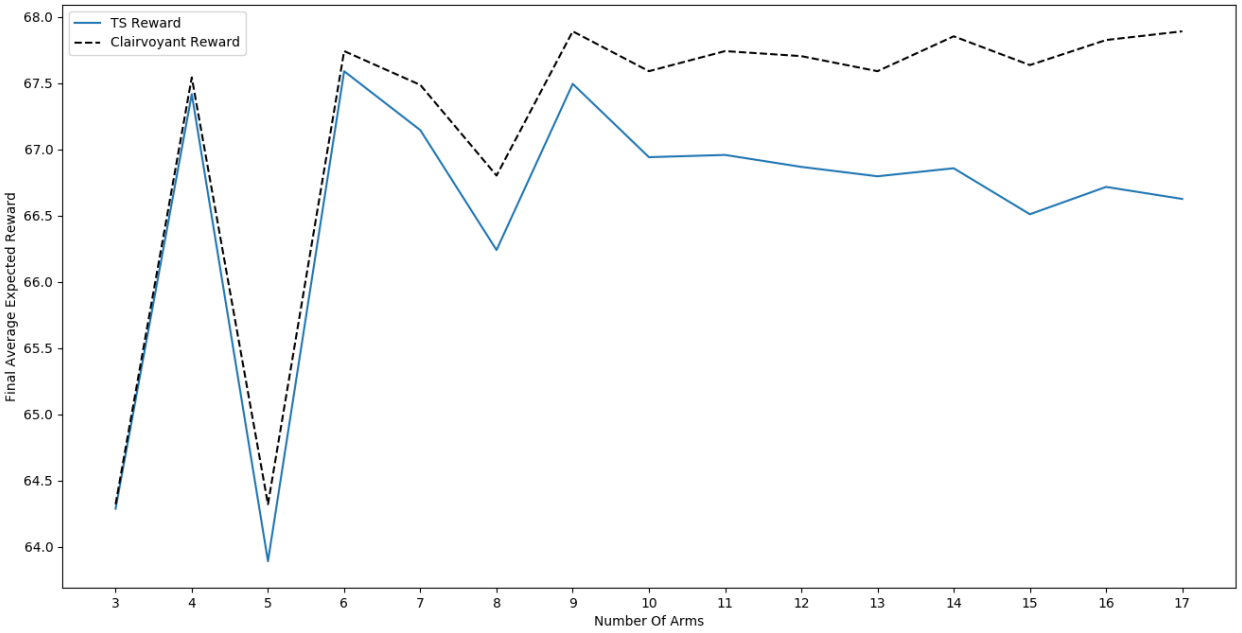

Figure 38: *Value of the average expected reward at the end of the optimization time, that is the time horizon, for each number of arms. The clairvoyant reward for each case is reported for reference.*

The considerations made so far are summarized in Fig. 38. The clairvoyant reward fluctuations are significant at first, slowly converging around the best candidate for the problem. At the same time, the distance between that value and the final expected reward slowly diverges. In a practical scenario, we might argue if it would really be worth considering 17 candidates instead of 9, for instance, as its clairvoyant arm is about the same, but convergence is much faster. However, since we do not possess such information, we can only decide based on how much are we willing to lose in order to enhance our probability to converge to a high-valued clairvoyant arm.

Finally, to complete the analysis, we show in Fig. 39 the cumulative regret over time

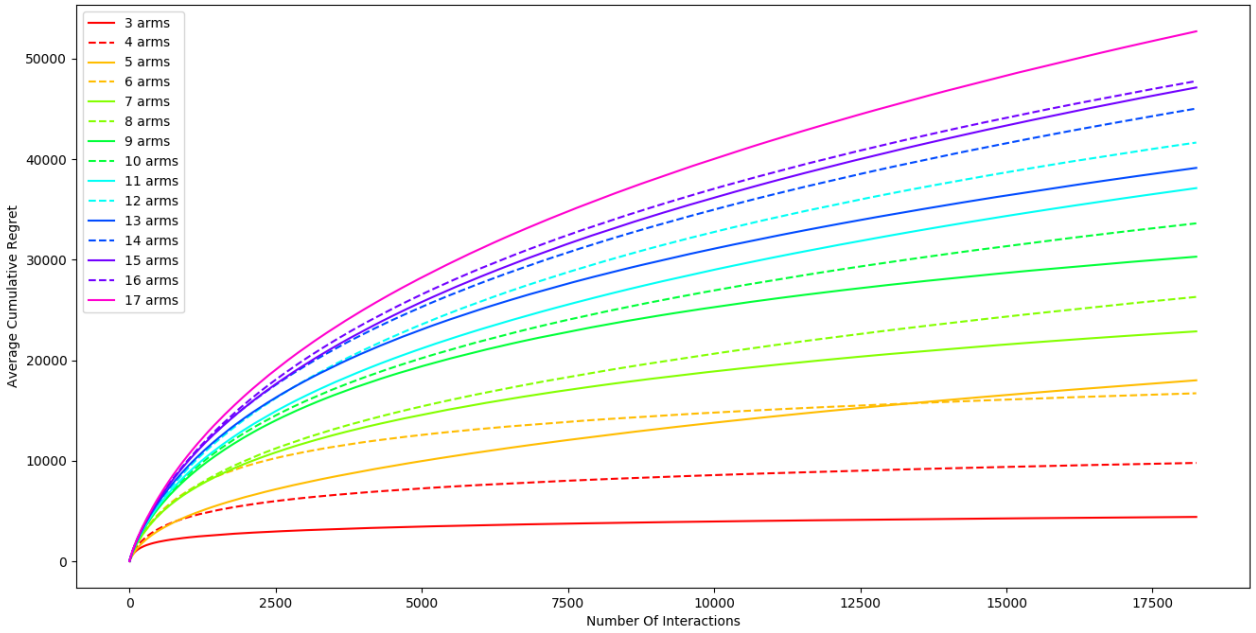

Figure 39: *Average cumulative regret growth as the number of interactions increases for a number of arms ranging from 3 to 17.*

considering different numbers of arms. The figure shows a clear correlation between number of arms and regret accumulated over time. For a number of arm higher or equal to 10, we might also argue that convergence has not been reached yet. However, we also notice some peculiar cases in which increasing by one the number of arms does in fact reduce regret. This behaviour can be observed in 5-Arms VS 6-Arms scenario and in 12-Arms VS 13-Arms scenario. This may be due to a variety of reasons, but the most likely is that the new disposition of candidates allows for a quick discrimination of the clairvoyant arm.
An important reminder is that regret, by itself, does not represent an indicator to choose between different number of arms in general. This is only true given that the clairvoyant reward stays the same. In fact, regret represents the efficiency of the process itself, given its number of arms and reward distributions. We will use this knowledge in the following section, where we use an ad-hoc defined metric to compare settings with a different number of arms.

Non-Stationary Environment

Considering a non-stationary environment introduces the problem of dealing with expired samples, that is, samples that were drawn before a phase shift occurred. If we have no information on how the arm reward distributions change over time (e.g. we cannot suppose that the difference between the old and new average reward for a given arm is upper bounded by a given value), we might find ourselves in the situation where an arm that had a very high reward before the phase shift has a very low reward after the shift. For this reason, any estimation we made before the shift is no longer reliable. However, discarding the whole set of samples gathered before the shift and starting anew (that is, supposing we can at least identify when the shift occurs) is a drastic choice, as we cannot be sure there is no correlation between the two phases, either. For this reason, a simple solution makes use of a sliding window, a sort of short-term memory that stores the latest n samples. This allows to slowly converge to the new clairvoyant arm by indirectly encouraging exploration when the samples of a given arm decrease.

In our experiments, we chose to analyze three different environments, respectively with 4, 10 and 17 arms, in order to understand how the number of arms impacts on the final results. Moreover, for each environment, we studied five different values for the sliding window. These values are discussed in Section 3. However, since there is no concept of bound for Thompson Sampling, we redefined SW_3 as:

$$SW_3 = \frac{T}{2 * F}$$

The fifth value for the sliding window is the number of interactions. This is equivalent to using no sliding window, such as in the stationary environment case.

$$SW_5 = T$$

We start by analyzing the average expected reward over the number of interactions for the different environments, shown in Fig. 40, 41 and 42. For quick reference, $SW_1 = 135$, SW_2 varies according to the learner, $SW_3 = 3041$, $SW_4 = 6083$ and $SW_5 = 18250$. As we can see, during the first phase, the highest expected reward is obtained by retaining all the information on the phase. In fact, the best performing learners use either SW_4 or SW_5. More precisely, we can even notice a direct proportionality between the size of the sliding window and the value of the expected reward at the end of the first phase. This is not surprising, since by discarding

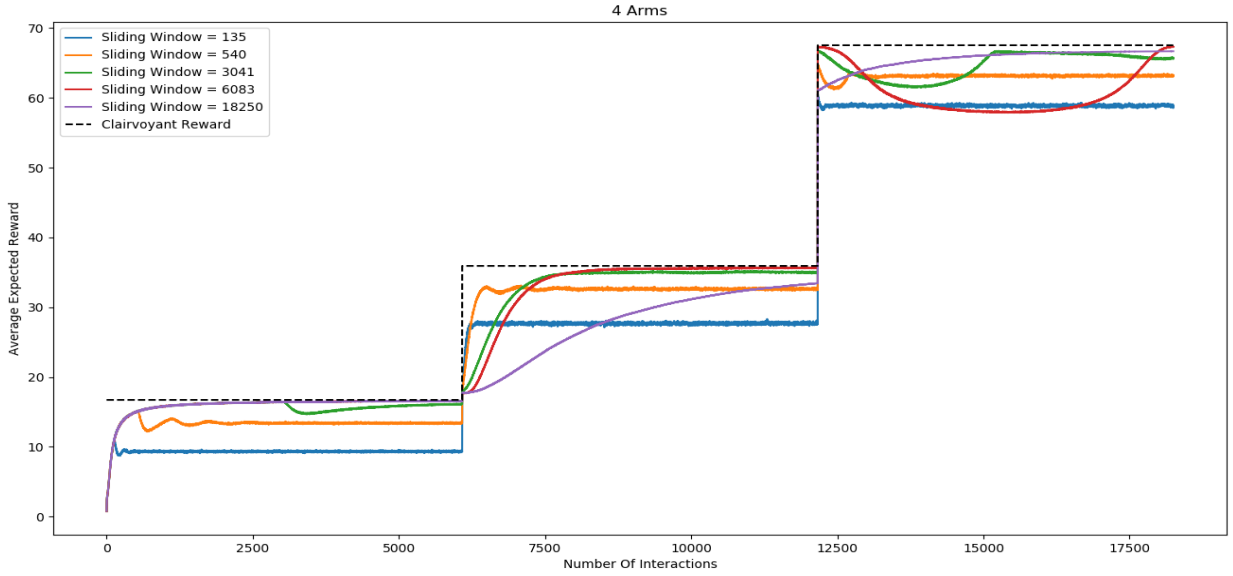

Figure 40: *Average expected reward growth per sliding window value as the number of interactions increases for an environment with 4 arms.*

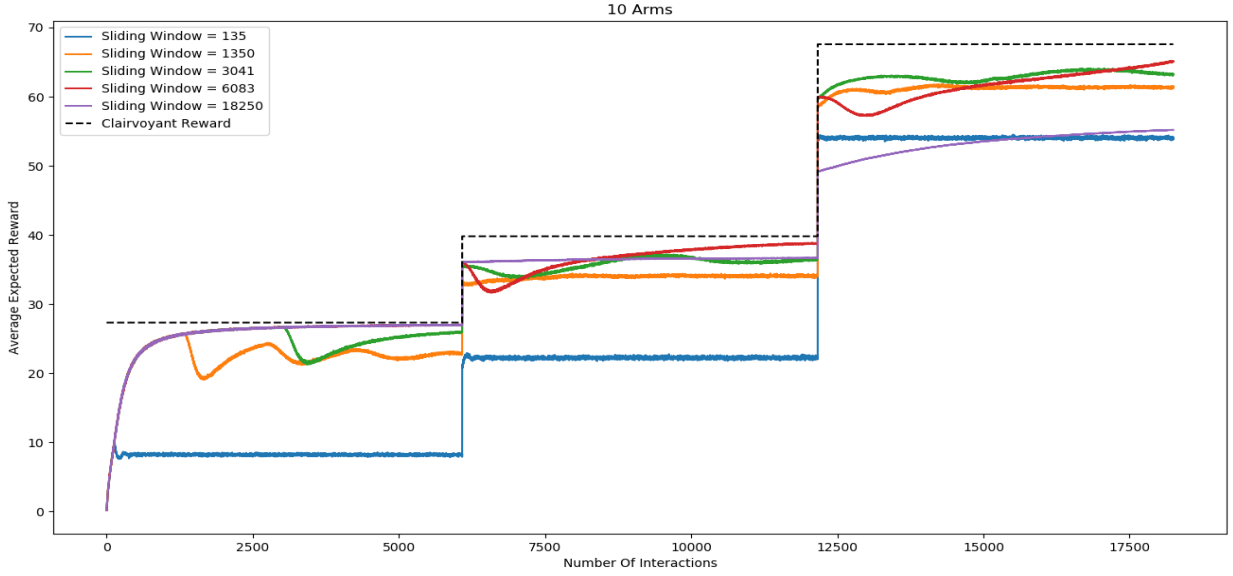

Figure 41: *Average expected reward growth per sliding window value as the number of interactions increases for an environment with 10 arms.*

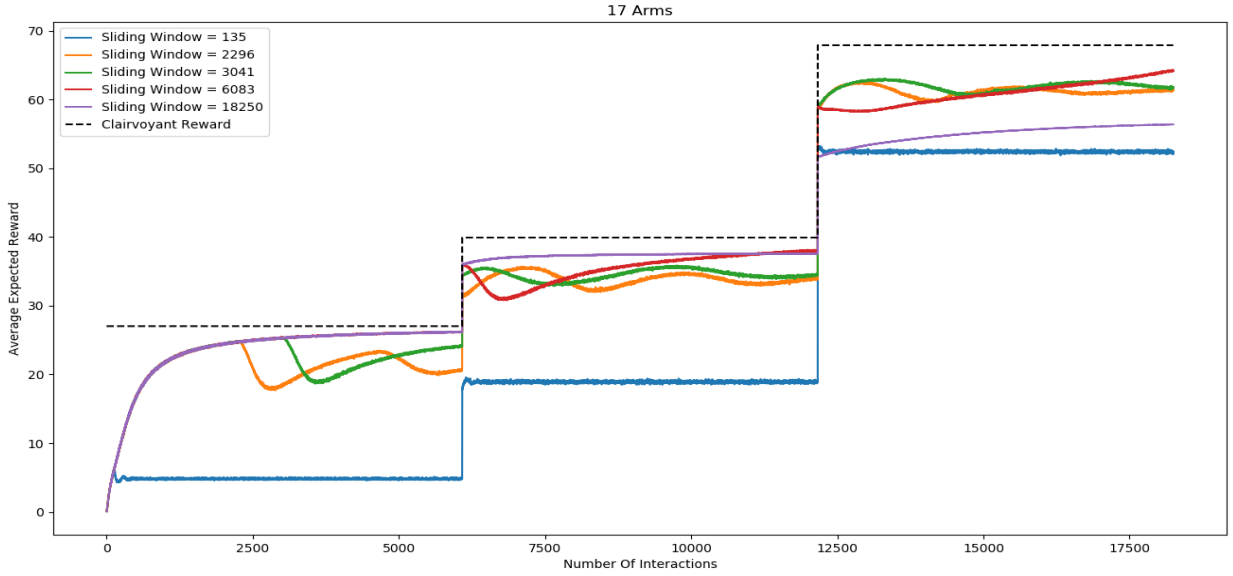

Figure 42: *Average expected reward growth per sliding window value as the number of interactions increases for an environment with 17 arms.*

samples during the phase we are sooner or later forced to explore old sub-optimal arms, inevitably accumulating regret. Another emerging trend is an oscillatory behaviour which slowly decreases in amplitude up to a minimum. These oscillations are centered around a given value of the expected reward which depends, once again, on the size of the sliding window: the larger it is, the higher the convergence value. More interestingly, convergence speed is related both to the length of the sliding window and to the number of arms considered: the learner with 4 arms was able to converge with all values of the sliding window, while convergence is still far both in the 10-arms and in the 17-arms learners for SW_2 and SW_3. This behaviour can be explained by looking what happens inside the sliding window. Up to being fully loaded, the learner behaves as if no sliding window was there at all. As old samples start to be discarded, it is still to early for the learner to realize that it is loosing memory, so it will continue pulling the optimal arm for a number of rounds depending on the reward associated to each arm. This explains the first peak value. As it does so, it starts discarding the first samples, that are in general related to a sort of initial exploration phase, as the learner was just starting to familiarize with the environment. For this reason, the learner will soon realize that it is loosing information. As a reaction, it will start pulling sub-optimal arms to collect the information back. A new exploration phase will start, which will probably cause, after a while,

the discard of samples related to the exploitation phase of the optimal arm. This loss of information on the optimal arm, combined with the necessity to explore, will generally advantage sub-optimal arms that are close to the optimal one. Even after identifying the optimal arm again, the learner will have less time to exploit it before starting a new exploration phase. Necessarily, the second peak will thus be lower than the first one. As this process repeats itself, oscillations will reduce in amplitude as the sliding window will cause the learner to acquire a periodic behaviour. At this point, in fact, it will be like if the sliding window was sliding over a predefined set of choices, with only a few small variations, due to which a full convergence will never occur. A clear example of this "bounded-convergence" behaviour is the learner using SW_1.

In the second and third phase, the general behaviour is the same, though the specific results depend on how rewards associated to each arm change over time. A very unusual case is the behaviour of the learner using SW_4 in an environment with 4 arms during the third phase (Fig. 40). As we can notice from Table 6 in Section 3, the optimal arm remains the same between the second and third phase. In fact, the average expected reward is already close to convergence at the beginning of the third phase. However, as a new exploration phase is triggered, the learner gets stuck on sub-optimal arms for quite some time. This is probably related to the excessive amount of information retained by the learner, which combined with the reward changes related to this specific case, produce a prolonged exploration. As a further proof, consider how the learner using SW_3 also experiences this prolonged exploration. However, thanks to its sliding window being half of SW_4, it is able to wipe the memory clean faster and, thus, converge faster. It is interesting to notice how the learner using SW_4 is still able to converge to a higher value.

We can now compare the efficiency of each learner by analyzing the average cumulative regret in Fig. 43, 44 and 45.

Since these plots show the regret accumulated by different learners in the same environment, we can actually compare their performance.

We immediately realize that SW_1 is the worst performing one. Its excessively reduced size forces a frequent exploration that brings to a fast increase in regret. Following, SW_2 and SW_5 show a final value of regret that is about the same for all number of arms. However, while SW_2 allows to maintain a sort of regular increase in regret, SW_5 is more efficient in the first phase, worsening in the second or third phase depending on the number of arms. The choices that lead to the best performances are thus SW_3 and SW_5. With the exception of an environment with 4 arms, where its performance, as we previously discussed, drastically worsens during the last phase, SW_4 seems to be best choice, even if SW_3 is still able to provide good

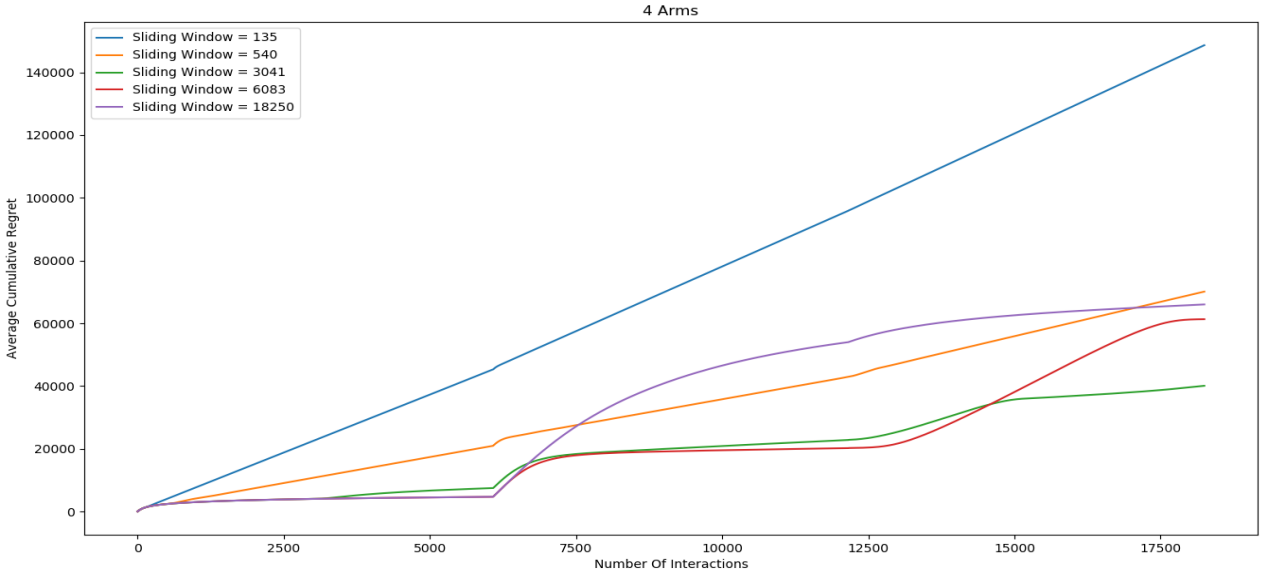

Figure 43: *Average cumulative regret growth per sliding window value as the number of interactions increases for an environment with 4 arms.*

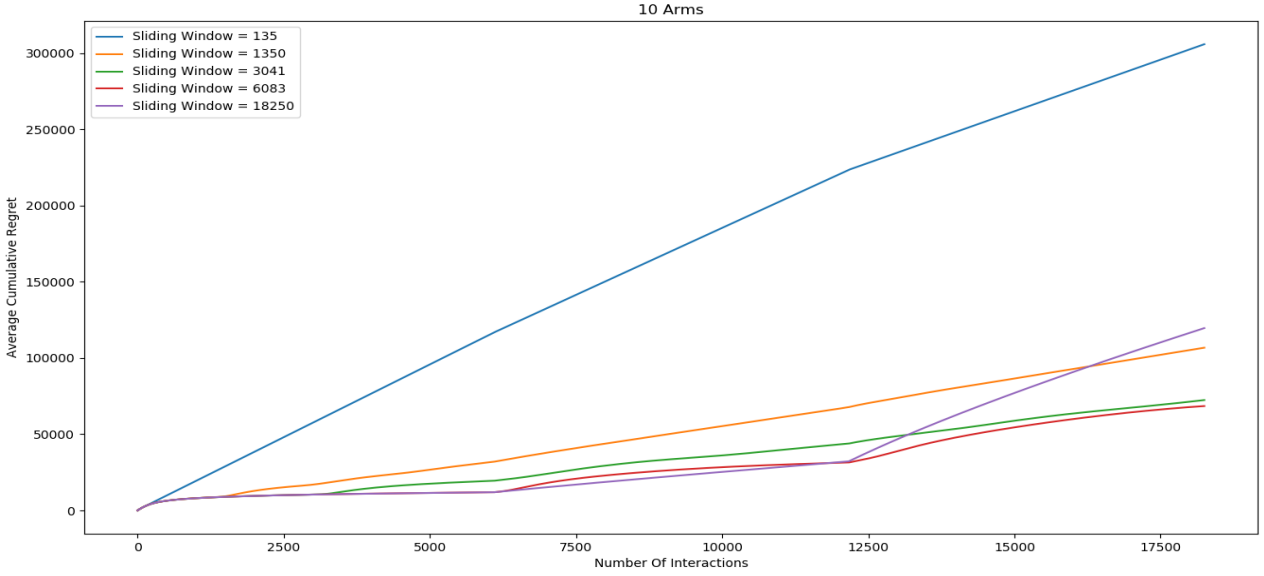

Figure 44: *Average cumulative regret growth per sliding window value as the number of interactions increases for an environment with 10 arms.*

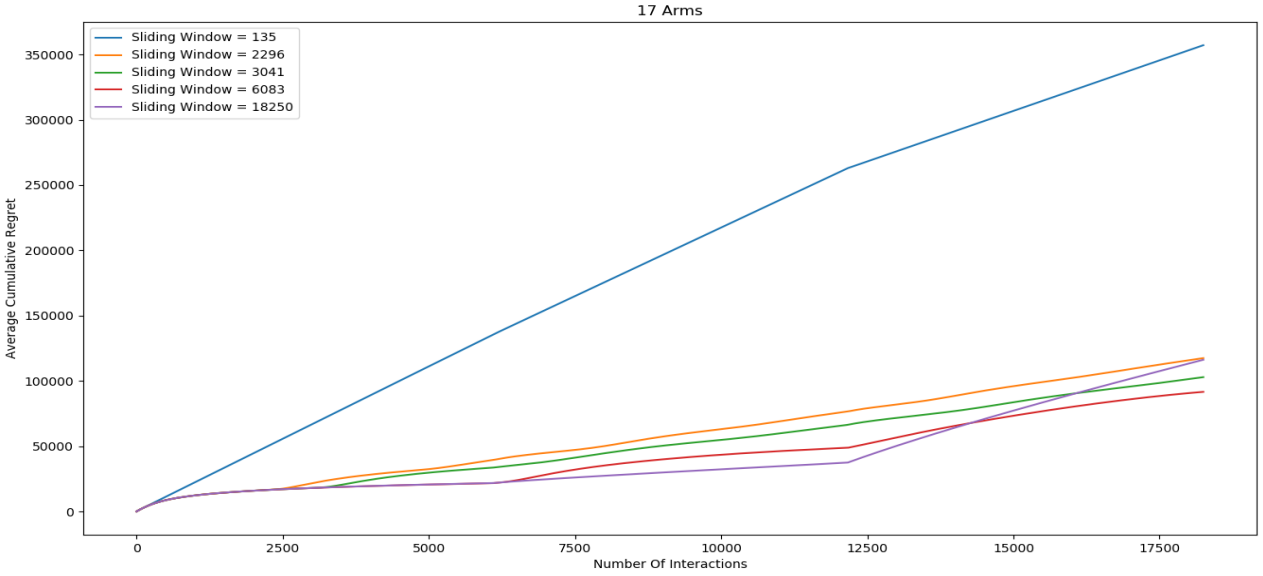

Figure 45: *Average cumulative regret growth per sliding window value as the number of interactions increases for an environment with 17 arms.*

performances.

To compare performances among learners with a different number of arms, we took the best two values of the sliding window, SW_3 and SW_4, and we compared, for each of them, the gain obtained using a learner with 10 and 17 arm with respect to that of a learner with 4 arms. In order to do so, we made use of the G-function defined at the beginning of this section. The results are shown in Fig. 46 and 47.

The intersection of the G-Function with the x axis represents the point from which using 10 or 17 arms becomes more profitable than just using 4. As we can see, this point comes very early in the plot, suggesting that using 4 arms with the given time horizon is generally a bad choice. On the other hand, a learner with 10 arms is able to gather much more reward compared to that with 17 arms, especially in the case of sliding window SW_3. With that in mind, by looking at previous Fig. 44, we can conclude that the best choice is SW_4 with 10 Arms.

Conclusion

To wrap up, we analyzed the performances of Thompson Sampling Algorithm in both stationary and non-stationary environments, with different number of arms

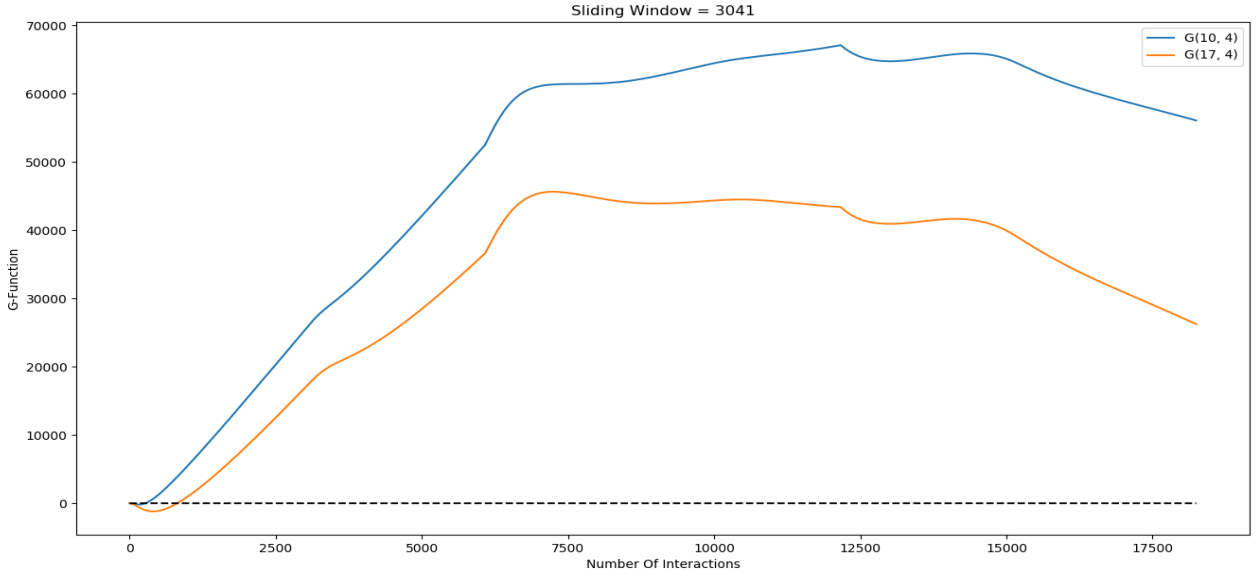

Figure 46: *G-function comparing the cumulative reward of two learners with 10 and 17 arms with respect to a learner using only 4 arms. The sliding window size is SW_3.*

Figure 47: *G-function comparing the cumulative reward of two learners with 10 and 17 arms with respect to a learner using only 4 arms. The sliding window size is SW_4.*

and values of the sliding window.

In the stationary case, the best choice a posteriori is using either 6 or 9 arms, depending on how important is the clairvoyant reward with respect to the expected reward obtained at the end of the process. The best choice a priori is, of course, using a number of arm that is closer to the maximum one that can be efficiently handled, which is 17, cause this increases the probability of finding the best price.

In the non-stationary case, for the value of the sliding window, we have to distinguish between two cases: non-availability of prior information and availability of prior information. If no prior information is available, then we can only choose between SW_1, SW_2 and SW_5. A posteriori, SW_1 is the worst performing with all number of arms considered. SW_5, which basically means using no sliding window, performs similarly to SW_2, as we can see from the cumulative regret analysis. However, this is a behaviour that is very specific to the case at hand. The general correct choice would thus be SW_3. This coincides with the best a priori choice, given that SW_1 is too small, while SW_5 is generally a bad choice on the long run. If prior information is available, we can use all values of sliding window we tested. In this case, SW_3 and SW_4 proved to be competitive choices, showing that even approximate knowledge of the phases length is a great advantage. Our analysis revealed that the best choice a posteriori is SW_3 for 3 arms and SW_4 for 10 and 17 arms. A priori, SW_4 is a more reasonable choice, since nothing suggests that using half the phase length might yield better results. For what concerns the number of arms, we can say that a posteriori the best choice is 10, given our G-function analysis. Moreover, by looking at tables 7 and 8 in Section 3, we can notice how close the profits related to the optimal arms per phases are. Thus, at the price of an increased regret, the choice of 17 arm does not bring major advantages. Similarly, the optimal arms per phase in the case of 4 arms are significantly smaller in the first two phases, which brings to a reduced cumulative reward on the long run. Again, a priori we still have better chances to converge to the best arms per phase by selecting 17 arms.

5.5 Performance Comparison

Here we present the result of the comparison of the different algorithms we analyzed.

Stationary Environment

In Fig. 48, we compare the expected reward convergence properties of each algorithm, while in Fig. 49 we compare their average cumulative regret, both at the end of the time horizon. As we can see, for a relatively high number of arms, K-Testing emerges as the one closer to convergence, followed by UCB1 and TS. However, it accumulates the largest regret, which strongly highlights its inefficiency. For UCB1 and TS, we should spend a few more words. As we can see, UCB1 shows an expected reward at convergence which is slightly closer to the optimal value, at the price of a slight increase in regret, barely visible in the plot. This is unlike what theory suggests, that is, that TS generally performs better than UCB1. However, it must be noted that we tuned the weight of the exploration term, that is, the bound of the UCB1 algorithm, through the parameter B. This made the algorithm more efficient, but it is an approach that is not always possible in practice, since it requires some previous knowledge of the environment. Without any information, modifying the weight of the bound is an arbitrary choice that can as well lead to a worsening in performance.

Finally, K-Testing shows a decrease in regret as the number of arms grows. This is because the optimization horizon is fixed, meaning that, as the number of arms increases, the number of samples drawn before choosing which candidate to keep decreases. As a result, given our configuration of candidates for the different number of arms, the times a very bad arm is pulled decreases over time, leading to a more homogeneous distribution of rewards for each sample. However, this is not a general case, since in a context where most candidates keep being very bad even for an increasing number of arms, regret might increase as well.

Non-Stationary Environment

This comparison only concerns UCB1 and TS, since K-Testing was not applied in a non-stationary environment. The following experiments are applied in a 10-arms environment using SW_4 as value of the sliding window. In Fig. 48 we show the expected reward of the two algorithms over the number of interactions. We also made use of the G-Function to provide a better comparison of the reward accumulated by the two algorithms.

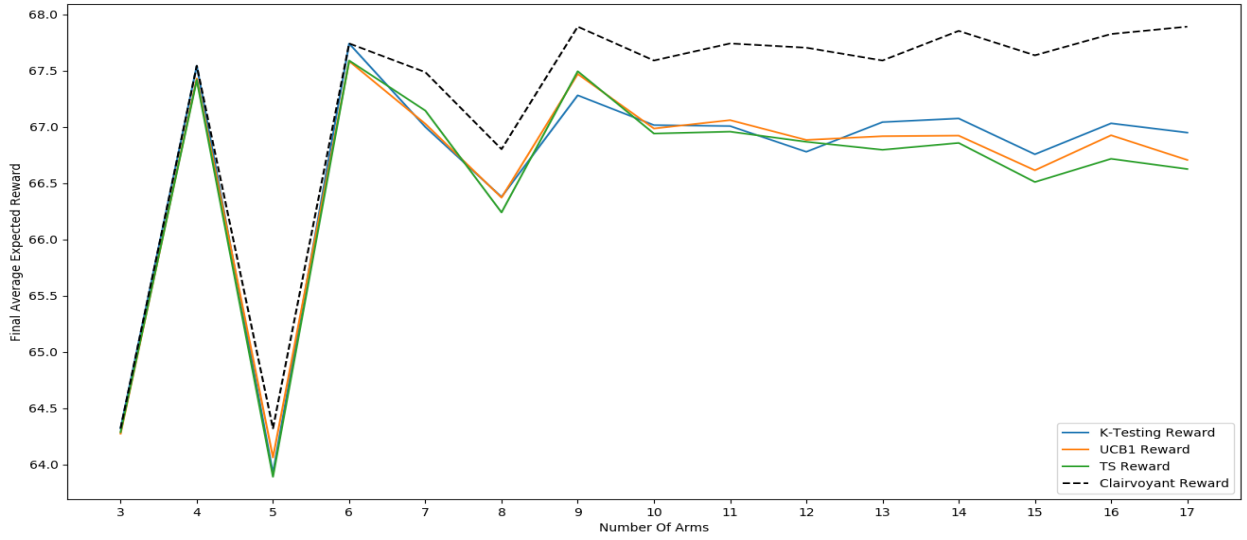

Figure 48: *Value of the average expected reward at the end of the optimization time, that is the time horizon, for each number of arms and each algorithm applied. The clairvoyant reward for each case is reported for reference.*

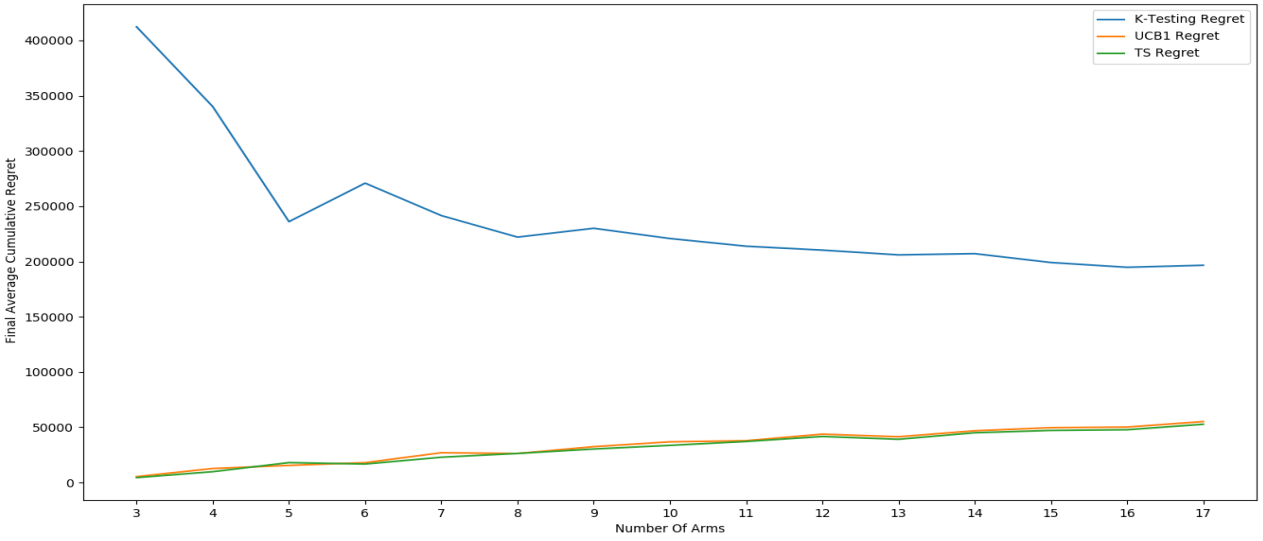

Figure 49: *Value of the average cumulative regret at the end of the optimization time, that is the time horizon, for each number of arms and each algorithm applied.*

TS is generally converging faster while simultaneously avoiding sharp falls of the expected reward at the beginning of a new phase. This is mainly related to the different nature behind the two algorithms. UCB1 is deterministic, that is, the arm to pull at the next iteration is always uniquely identified by the samples drawn up to that point. On the other hand, TS is probabilistic, that is, the arm to pull is chosen by comparing samples drawn from probability distribution related to each arm. For this reason, the next arm to pull can never be known a priori with full confidence. In practice, UCB1 will generally be forced to pull sub-optimal arms at the beginning of a new phase, because it will start losing the few samples that were drawn from them before converging to the optimal one. Also, when no more samples are available for one arm, UCB1 is forced to choose those arm to be able to compute its bound. TS, on the other hand, does not show this sharp changes, as reducing the number of samples just makes more probable that one arm will be pulled again, without making it an obligation. This also makes TS able to work with arms it has no samples of, without forcefully pulling them when all samples are discarded. From the G-function, we notice that the extra regret accumulated by TS with respect to UCB1 grows almost monotonically, decreasing only in the last phase.

To complete the analysis, in Fig. 52, we compare the regret accumulated by the two algorithms, which confirms the higher efficiency of TS over UCB1.

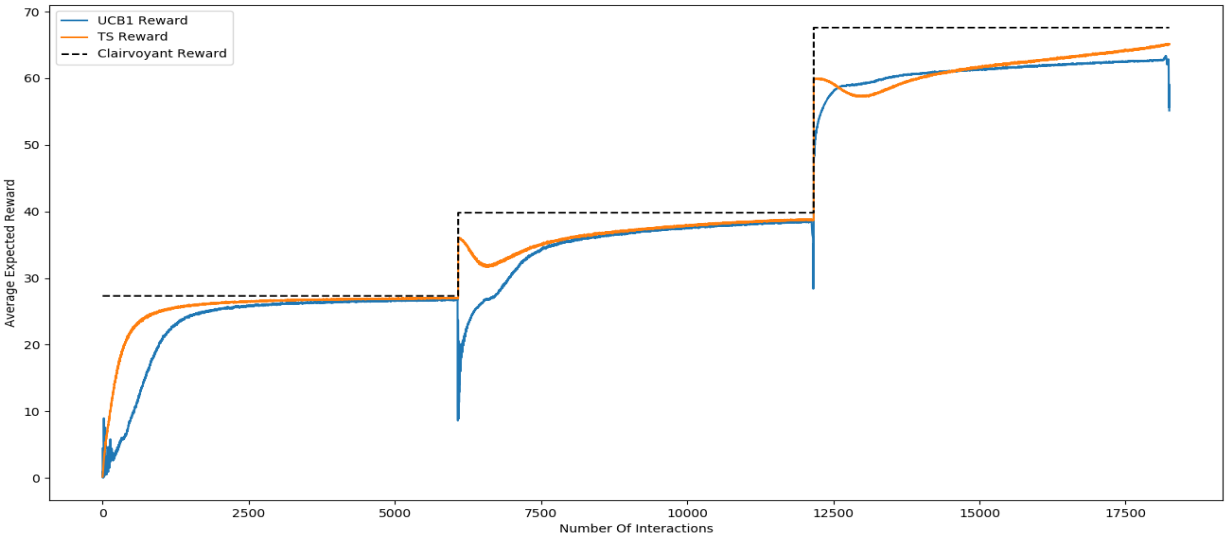

Figure 50: *Average expected reward convergence as the number of interactions increases for two learners respectively using UCB1 and TS. The number of arms considered is 10.*

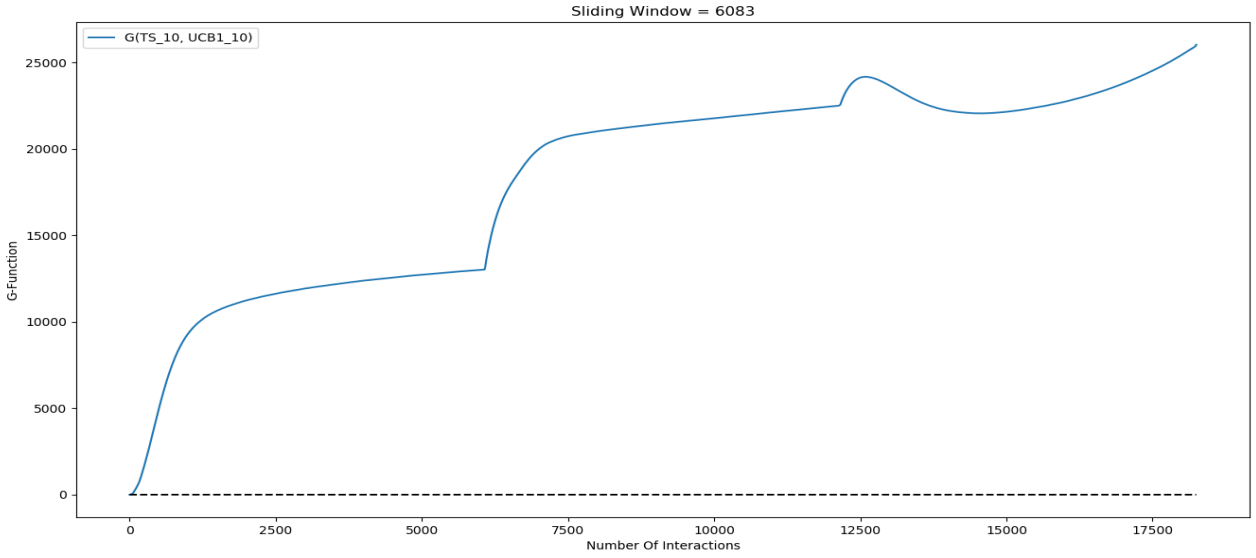

Figure 51: *G-function comparing the cumulative reward of a learner using TS with respect to a learner using UCB1, both considering 10 arms. The sliding window size is* SW_4.

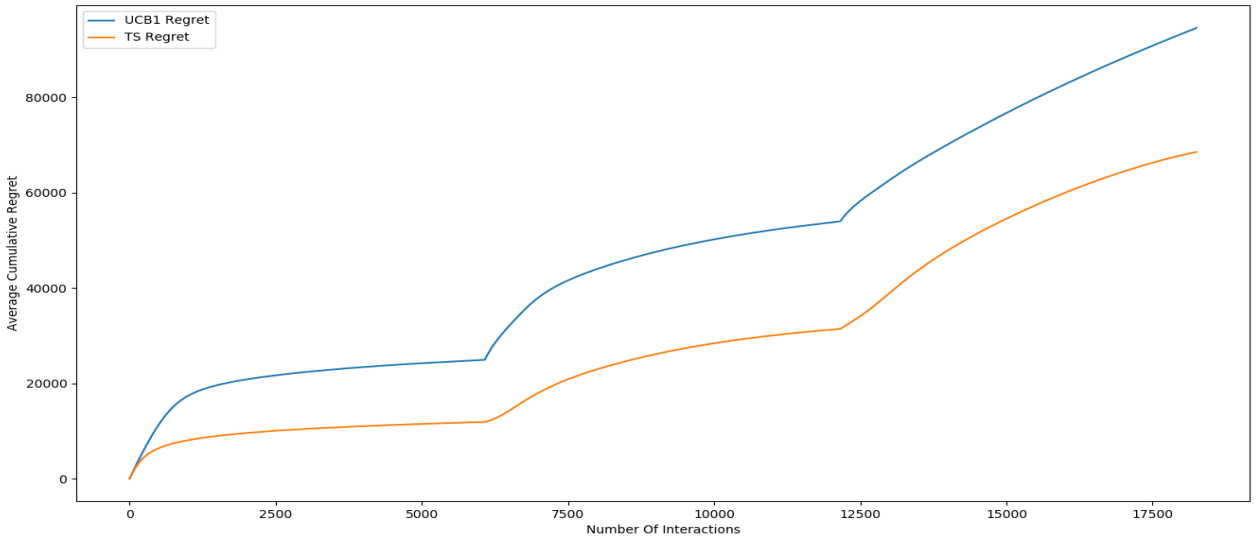

Figure 52: *Average cumulative regret growth as the number of interactions increases for two learners respectively using UCB1 and TS. The number of arms considered is 10.*

6 Optimization with splits

Performing context generation means trying to understand the different behavior that different types of customers follow. Since we are considering the case of an online shop, this can be useful to perform price discrimination and further increase the reward.

In our experiments, we consider the three classes of users that are described in Section 3.1 and we want to understand whether it is useful to divide the users in the identified classes and apply price discrimination or not.

Since we have 50 observations per day and we need to check the eventual usefulness of introducing the splits every Monday, this check is done every 350 observations.

6.1 Context Generation with UCB1

Stationary Environment

In the case of stationary environment, we will only consider the *high interest phase* with no sliding window. Figure 53 reports the expected reward achieved by the

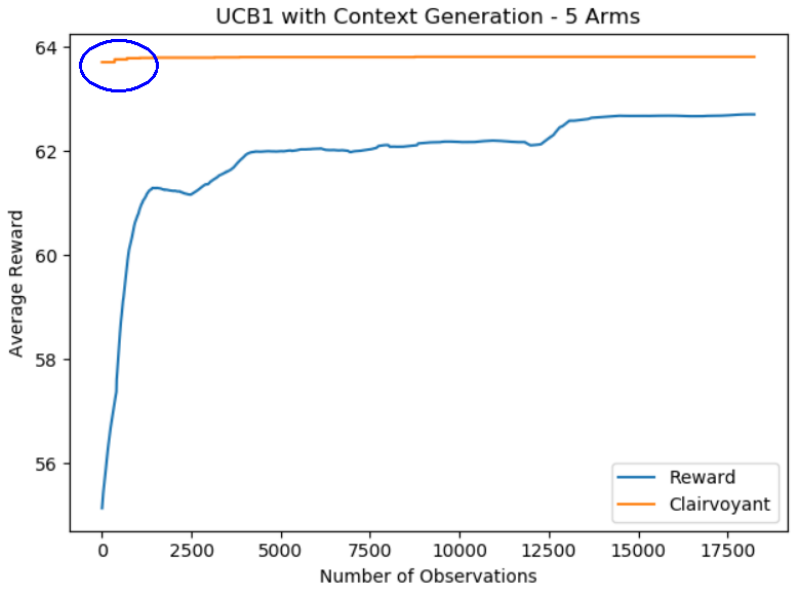

Figure 53: *Expected reward achieved by the UCB1 algorithm when performing context generation in a stationary environment with 5 arms.*

UCB1 algorithm when performing context generation in a stationary environment.

The orange curve is the *average* over several experiments of the clairvoyant, while the blue line is the average reward. The arms that were explored in this case are those reported in Table 3. It is worth noticing that the splits happen after very few weeks, as there are only *5 Arms* to explore. Some frequent final **decision trees** of the price discrimination are described in Figure 54.

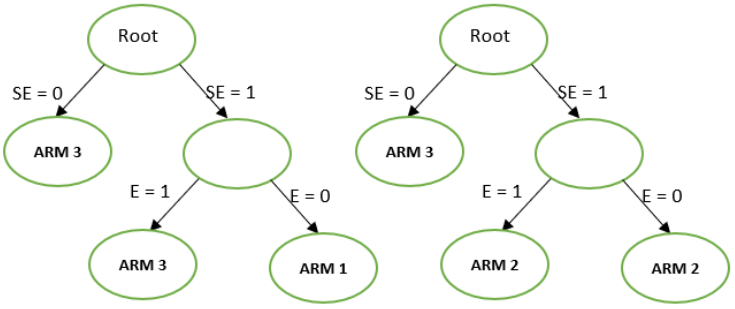

Figure 54: *Frequent final **decision trees** of the price discrimination when running UCB1 with context generation on the stationary environment with 5 arms.*

In the picture, SE stands for **Southern Europe** and E stands for **Employed**. The decision tree is not always the same because the process involves some randomness and it would require more time than the allowed time horizon to converge (as the Arm that is chosen for each class can always change when the confidence bounds shrink). Furthermore, after splitting the users into classes the amount of samples per class is reduced, making the process potentially converge slower than UCB1 without context generation.

The same study was carried out for the cases with 10 and 17 arms.

Figure 55 shows the performances achieved by the UCB1 algorithm when performing context generation in a stationary environment with 10 arms. The arms used in this experiment are the ones listed in Table 4. In this case, as there are more arms to explore, the splits do not always happen at the beginning of the year. In fact, the orange line represents the average clairvoyant's expected reward and we can see it keeps making little steps (which correspond to splits that yield to little improvements) even at almost half of the year.

Figure 56 shows some frequent final decision trees.

The same reasoning applies to the case with all the 17 arms, whose results are reported in Figure 57 and Figure 58.

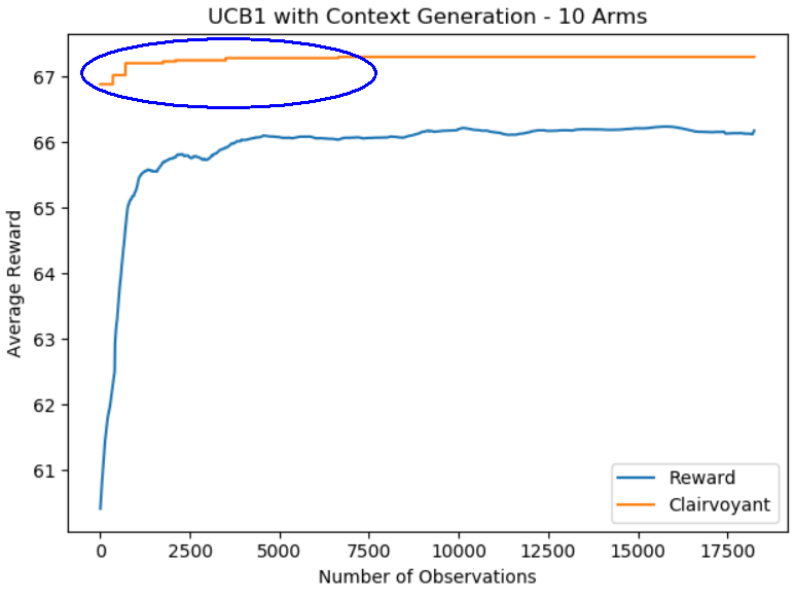

Figure 55: *Expected reward achieved by the UCB1 algorithm when performing context generation in a stationary environment with 10 arms.*

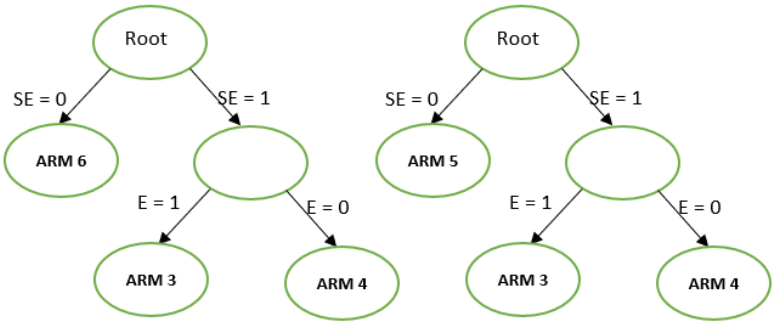

Figure 56: *Frequent final* ***decision trees*** *of the price discrimination when running UCB1 with context generation on the stationary environment with 10 arms.*

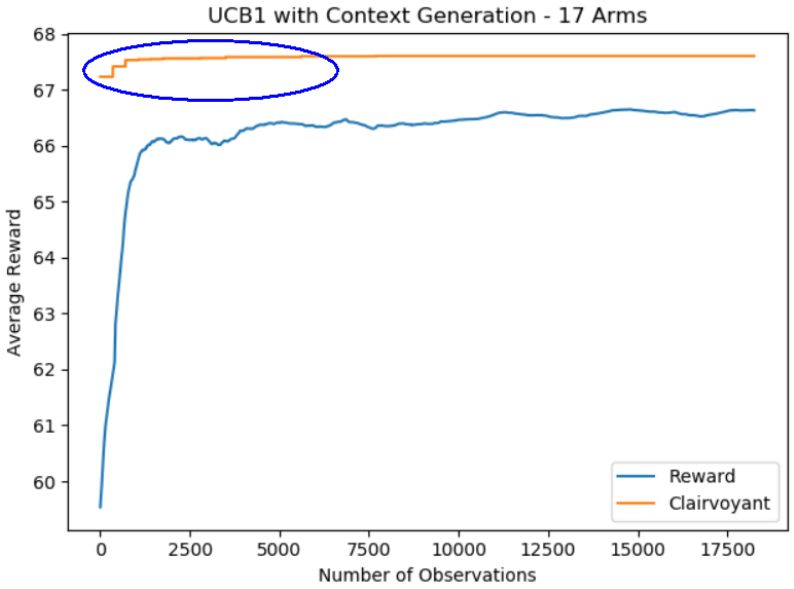

Figure 57: *Expected reward achieved by the UCB1 algorithm when performing context generation in a stationary environment with 17 arms.*

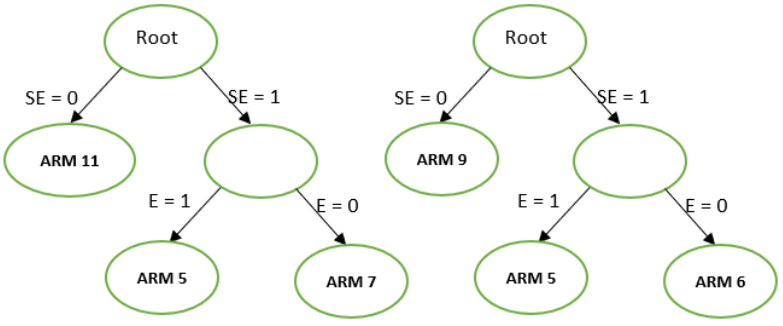

Figure 58: *Frequent final* ***decision trees*** *of the price discrimination when running UCB1 with context generation on the stationary environment with 10 arms.*

Non-Stationary Environment

In this section, we analyze the performances of the UCB1-based Context Generation procedure in a non-stationary environment in the two scenarios in which a sliding window is used or not.

In all the experiments, we assume to begin the process in the *low interest phase*, and then move to the *medium interest phase* and subsequently to the *high interest phase*. In the case of non-stationary environment, the best arm for each category of users is different from phase to phase, therefore just representing the final one would not make any sense.

The first experiment shows the performances of the procedure on the case of non-stationary environment with no sliding window.

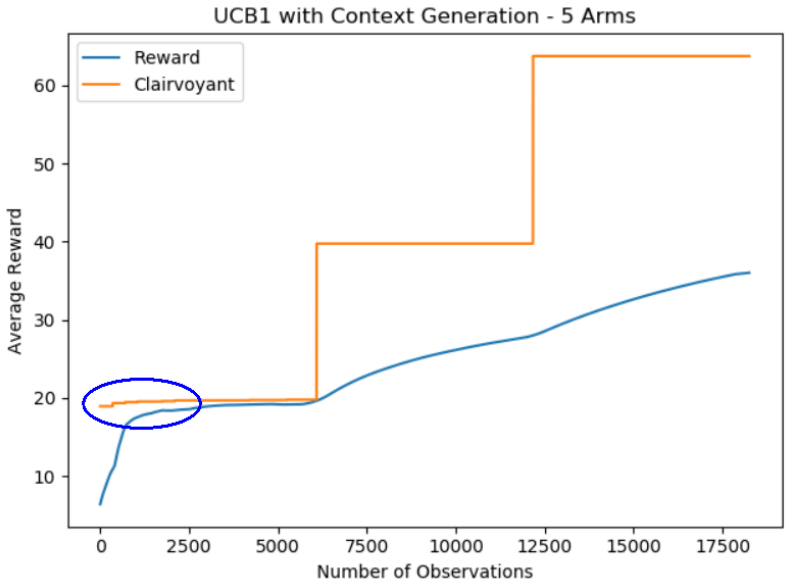

Figure 59: *Expected reward achieved by the UCB1 algorithm when performing context generation in a non-stationary environment with 5 arms and no sliding window.*

From Figure 59, Figure 60 and Figure 61 we can see how the algorithm that has no sliding window fails to adapt to the environment changes. It is worth noticing that in all the cases the users are divided into classes during the first phase. From the moment the users are divided in classes they are never merged again (even after the environment goes through abrupt changes).

The second experiment focuses on evaluating the performances of the algorithm on a non-stationary environment when a sliding window of 3000 observations is used.

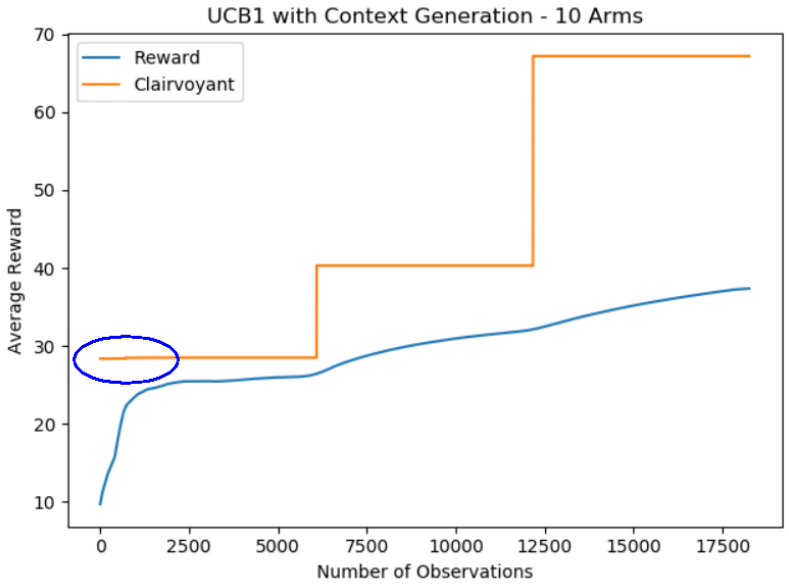

Figure 60: *Expected reward achieved by the UCB1 algorithm when performing context generation in a non-stationary environment with 10 arms and no sliding window.*

As there are 3 phases and 6084 observations are made during each phase, a sliding window of 3000 allows to completely "forget" the observations made in the previous phase from the half of the current phase. The results of this experiment are shown in Figure 62, Figure 63 and Figure 64.

It is noticeable that in all the cases (5, 10 and 17 Arms) the algorithm tends to reach rewards that are reasonably close to the clairvoyant, representing the maximum possible reward with the current split. This means that the algorithm is not always pulling the best arm and instead it keeps exploring other arms from time to time. The cause of this is the slow convergence of the algorithm united with the presence of the sliding window itself, that somehow sets a limit to how small the confidence bounds can be and allows them to become larger again as soon as old samples are forgotten.

Different values of the sliding window size can be tested. However, increasing the size of the sliding window may not be as beneficial as expected. In fact, there are two factors that come into play:

- a larger window size will cause the variables in the current phase to be affected by the past phase for a larger period of time. This may cause the reward to be lower at least until the past phase is completely forgotten;
- conversely, after the past phase is completely forgotten, the average reward

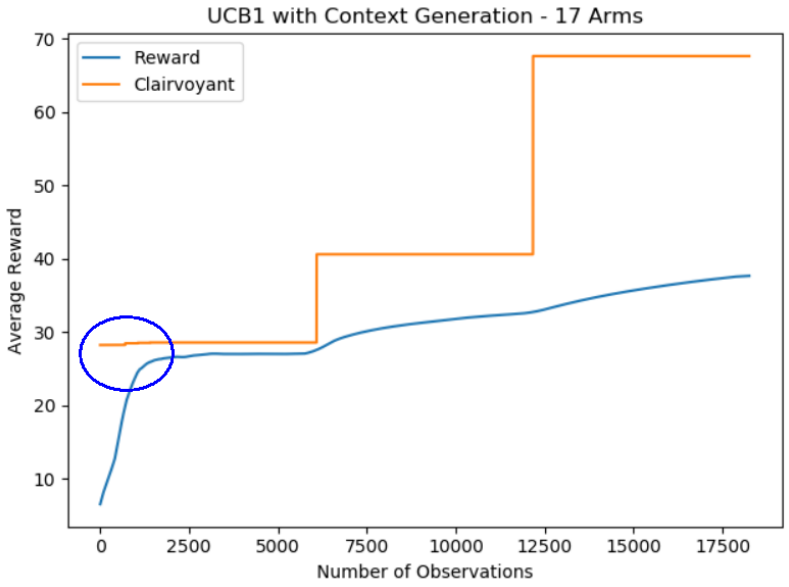

Figure 61: *Expected reward achieved by the UCB1 algorithm when performing context generation in a non-stationary environment with 17 arms and no sliding window.*

may be higher.

Given those two factors, it is difficult to tell a priori whether a larger window size will improve the results or not. A possibly good choice would be choosing the window size according to the number of arms, as explained in Section 4.

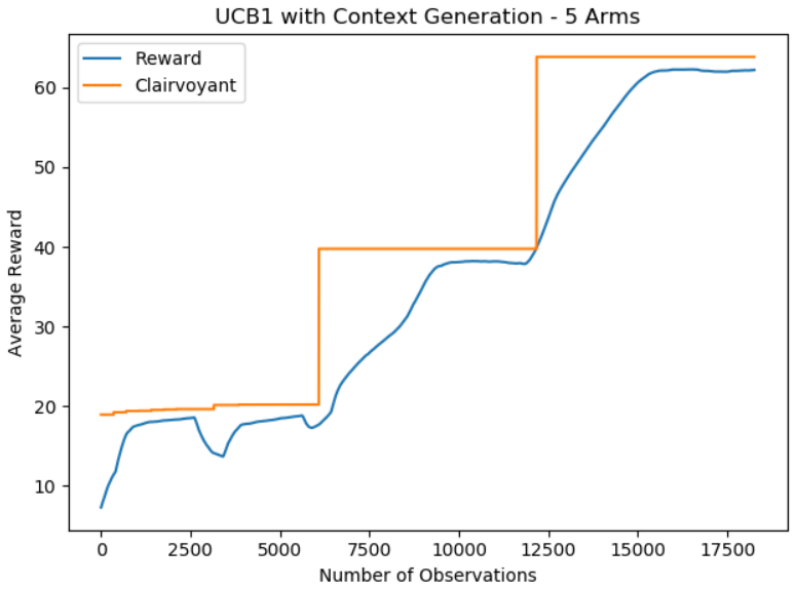

Figure 62: *Expected reward achieved by the UCB1 algorithm when performing context generation in a non-stationary environment with 5 arms and a sliding window of 3000 observations.*

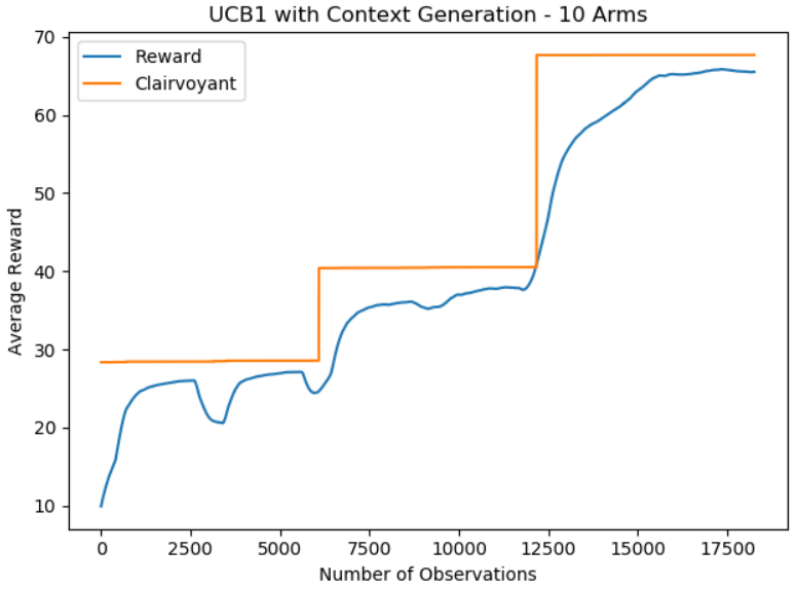

Figure 63: *Expected reward achieved by the UCB1 algorithm when performing context generation in a non-stationary environment with 10 arms and a sliding window of 3000 observations.*

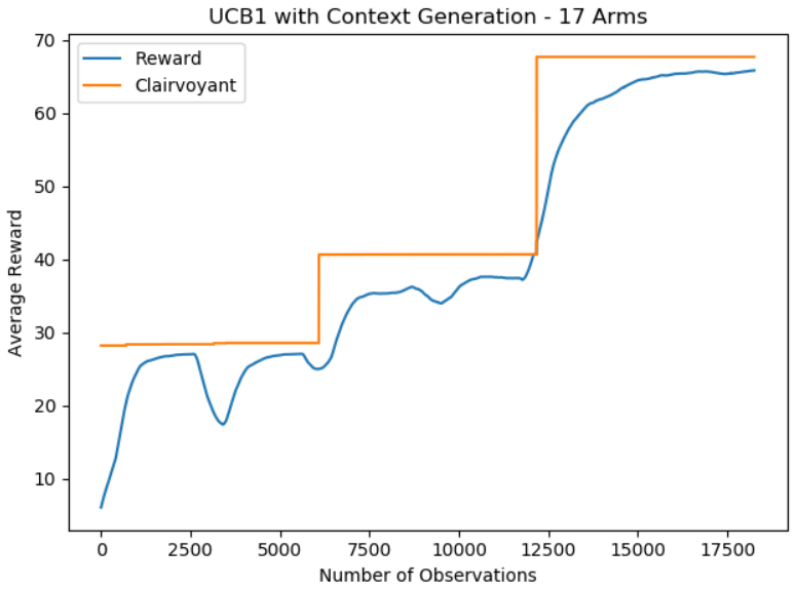

Figure 64: *Expected reward achieved by the UCB1 algorithm when performing context generation in a non-stationary environment with 17 arms and a sliding window of 3000 observations.*

6.2 Context Generation with Thompson Sampling

Non-Stationary Environment

In this section, we analyze the performances of the TS-based Context Generation procedure in a non-stationary environment in the two scenarios in which a sliding window is used or not.

In all the experiments, we assume to begin the process in the *low interest phase*, and then move to the *medium interest phase* and subsequently to the *high interest phase*. In the case of non-stationary environment, the best arm for each category of users is different from phase to phase, therefore just representing the final one would not make any sense.

For the sake of brevity, we will directly compare the performances of this procedure with the results obtained by the UCB1-based Context Generation procedure.

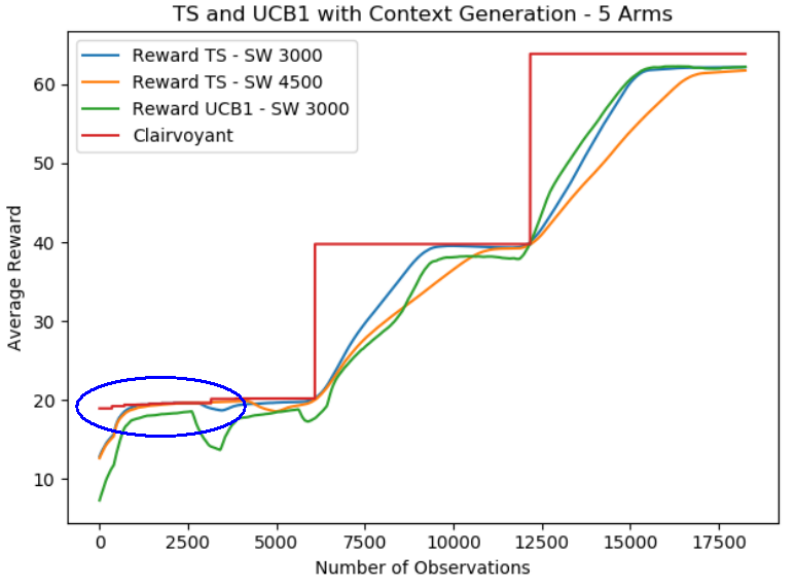

Figure 65: *Expected reward achieved by the TS and UCB1 algorithms when performing context generation in a non-stationary environment with 5 arms.*

Figure 65 compares the two approaches in the case with 5 arms. From the figure we can notice that UCB1 is superior to TS in the case with a sliding window of 3000 samples. Furthermore, increasing the size of the window to 4500 did not bring any improvement to TS. In fact, the reward in the case with SW=3000 increases faster and the reward in the case with SW=4500 does not get higher that the former even after all the samples of the previous phase are forgotten. As for all the other

experiments, the splits always happen before the middle of the first phase, and from that point the classes are never merged again.

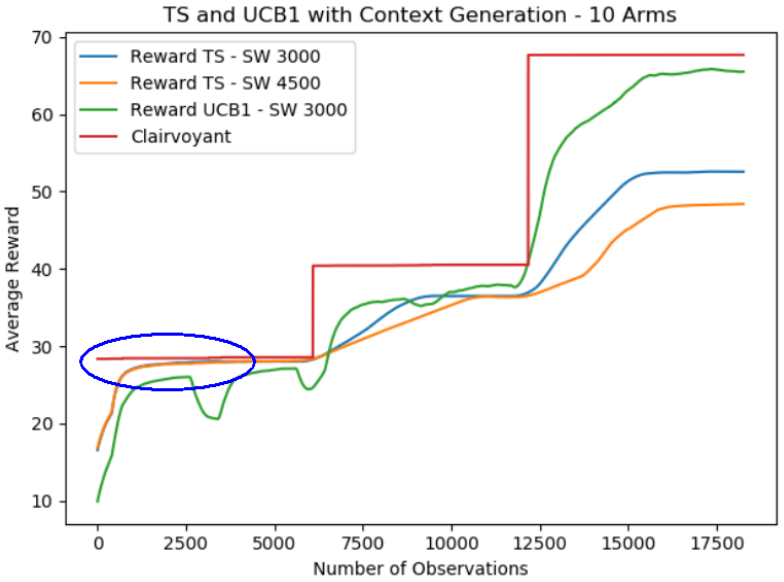

Figure 66: *Expected reward achieved by the TS and UCB1 algorithms when performing context generation in a non-stationary environment with 10 arms.*

Figure 66 compares the two approaches in the case with 10 arms. From the figure we can notice that UCB1 is superior to TS in the case with a sliding window of 3000 samples and that increasing the size of the window to 4500 did not bring any improvement to TS.

Similar reasonings apply to the case with 17 arms, showed in Figure 67.

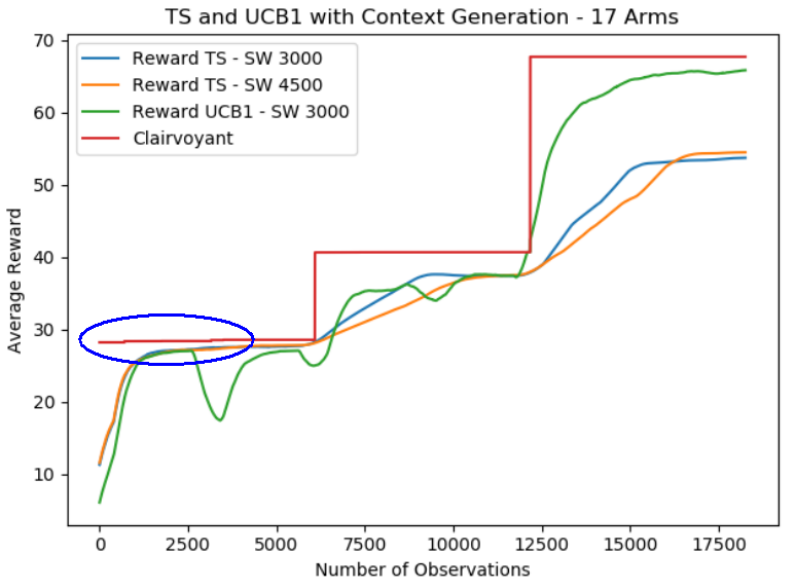

Figure 67: *Expected reward achieved by the TS and UCB1 algorithms when performing context generation in a non-stationary environment with 17 arms.*