Data Intelligence Applications - 2019/2020

Luca Alessandrelli, Jacopo Pio Gargano, Riccardo Poiani, Tang-Tang Zhou
Politecnico di Milano

July 1, 2020

Contents

1 Introduction

Among the presented possibilities, we have chosen to carry out the project related to pricing combined with advertising. The document is organized in the following way: we begin with an introductory chapter in order to define all the ingredients of the project (e.g. product, user features); after that, for each of the requested points, one chapter is present, explaining the problem, the plots, and the followed approach.

1.1 Product and users descriptions

During these hard times, we propose ourselves to study the problem of selling an off-brand hand sanitizer. More precisely, we consider the possibility of selling it in a pack of 6 items.
We imagine to advertise interested users via Facebook and we claim that we can observe the two following binary features:

- Medical interest: these users has shown during their interaction with Facebook a particular interest toward health and wellness. This is an obvious reasonable feature of interest
- Click often: users who clicks on ads, in general, very often

We believe these features to be practical since Facebook may estimate them reasonably well, allowing us to exploit them while pricing the users.
Sub-campaigns are targeted towards three different user contexts:

		Medical Interest	
		False	True
Click Often	False	C_3	C_1
	True	C_1	C_2

Table 1: Considered users contexts.

1.2 Scenario description

In order to use data distributions that resemble somehow the "real" ones we carried out some research. In particular, our interest aimed for two quantities:

- The cost per click (CPC), for what concerns the advertising aspect of the problem. Indeed, once we are aware of this quantity, we can estimate the number of visits to the website as a function of the budget. We contacted an expert in the advertising field to obtain a reasonable estimate of real numbers: we believe that a cost per click of $0.3 €$ is a large upper-bound on the quantity.
- The conversion rate probability (CRP), to model the users' behavior once they visit the website. We posit that an extremely under-priced item w.r.t. to the market can be sold with an high rate (e.g. 20% to very interested users). Moreover, for the considered good category, a descending CRP as a function of the price seems to be an obvious choice.

Generally speaking, we conjecture to sell the good in a price range of $[15, 25]$ euros. The market price is around $20 €$ for the proposed product. A fixed cost of $12 €$ is considered, for the profit maximization problem. A cumulative budget of $1000 €$ is considered available for each

day. Note that this total budget is shared among different sub-campaigns.
We study the behaviors of our algorithms under different data distributions, that are presented as scenarios: in particular, the conversion rate probabilities, the expected profit per user, and the distribution of the obtained number of clicks are presented for each context in each scenario. Additional plots, will be presented throughout the other chapters, when further explanations are needed.

1.2.1 Scenario: linear price and linear clicks

In this first scenario, we consider linear functions (in expectation) both for the pricing problem, and for the number of clicks obtained with advertising: curves are presented in Figure 1.
As shown in the picture, a Bernoulli random variable is used to model the CRPs: the mean of these variables are the one reported. As already pointed out, a descending behavior as a function of the price is used to model the user purchases. The expected profit, given a user context, is computed multiplying the respective CRP with the profit related to that price (i.e. $price - cost$). Note that due to the immediate interpretability of the proposed user feature, context 2 is reasonably the ones whose provide more revenue, while context 3, whose are the less sensitive users, are the ones who provide less profit.
For what concerns the number of clicks, samples come from a Gaussian distribution whose mean is shown in the picture, and whose standard deviation is constant to 25 for each budget value and each user context. Moreover, when a budget of 0 euros is used, no samples are generated, and the number of obtained clicks is simply 0. The saturation has been applied since it is reasonable that after that a certain numbers of daily users has been reached, no more users will be available for clicking the ads. However, this effect has not been considered for context 3, since these users (who miss both features) are believed to be in a larger number. Moreover, we might notice as context 1 saturates later than context 2, since users who shows only one of the two feature (see Table 1) are believed to be in a larger number w.r.t. the ones who have both features.
In this scenario a single phase is considered, and somehow, it relates to a normal market situation.

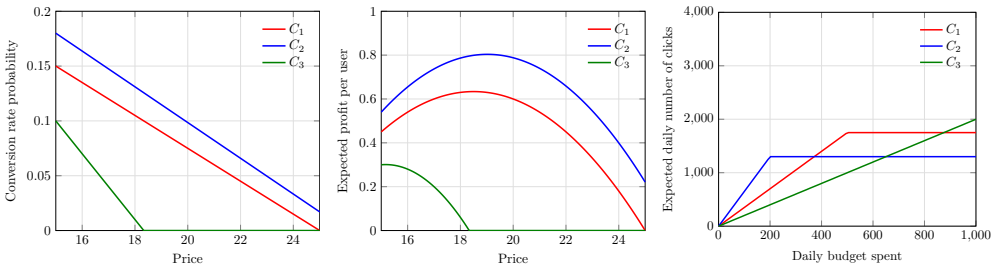

Figure 1: (left) Scenario 1.2.1 conversion rate probabilities per user context. (middle) Scenario 1.2.1 expected profit per user context. (right) Scenario 1.2.1 expected daily number of visits per daily budget spent and user context.

1.2.2 Scenario: tanh price and linear clicks

The second scenario is very similar to the first one. The only change stands in the conversion rate probability function: we overcome the linearity of the CRPs function, by means of tanh functions. CRPs and expected profit functions are reported in Figure 2. With the conversion rate probability of the first class, we model the behavior that the most interested users will switch

automatically to the competitors, when the proposed price is higher w.r.t. the market one.

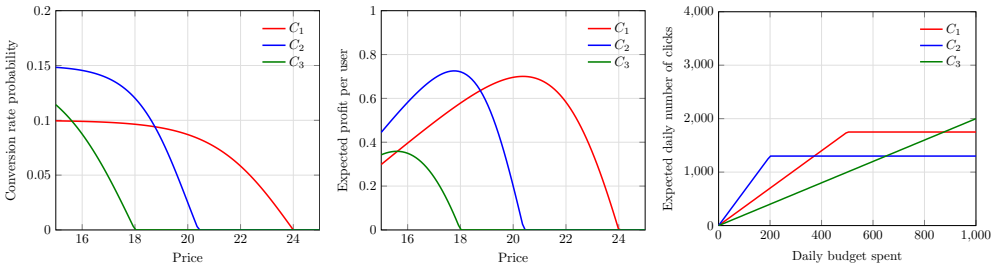

Figure 2: (left) Scenario 1.2.2 conversion rate probabilities per user context. (middle) Scenario 1.2.2 expected profit per user context. (right) Scenario 1.2.2 expected daily number of visits per daily budget spent and user context.

1.2.3 Scenario: non-stationary number of clicks

This scenario, shown in Figure 3 concerns 3 abrupt phases, that makes the number of clicks change dramatically. CRPs functions are not reported since this scenario is considered only from a number of visit maximization approach.
In particular, the 3 phases have the following meaning: in the beginning, a situation of normal market is presented; then, it follows a sanitary emergency situation; and, in the end, the post-emergency circumstances are presented.
We can note that during the emergency, the interest of all the users increases, and, also the saturation gets shifted. In the last phase, we conjecture that the usually uninterested users, increase significantly, since they are in a larger number and they might be more concerned of a possible crisis relapse.

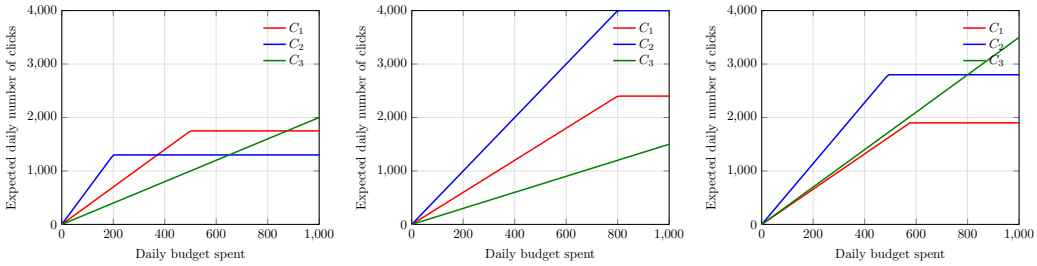

Figure 3: (left) Scenario 1.2.3 number of clicks during a normal market phase. (middle) Scenario 1.2.3 number of clicks during a sanitary emergency. (right) Scenario 1.2.3 number of clicks right after a sanitary emergency is over.

2 Stationary advertising

2.1 Problem description

We optimize the budget allocation over the three sub-campaigns to maximize the total number of user clicks on our advertisement through a combinatorial bandit algorithm. A unique phase is considered in this case (see Section 3 for the non-stationary case). Moreover, we assume that the performance of every sub-campaign is independent of the performance of the other sub-campaigns. This is not true in general, but it is a reasonable assumption for the considered purpose and scenario.

2.2 Algorithm design choices

We design a bandit algorithm to tackle the aforementioned problem. In general, in a bandit setting we have a set of choices, represented by the "arms" of the bandit. When we pull an arm we observe a reward. The objective is to understand which is the arm with the highest reward and, therefore, the one to go for. Generally, the arms are independent from one another. In this case, instead, the arms are correlated and information retrieved for an arm provides information also for the reward of the arms close to it. Therefore, we shall exploit this property.
Moreover, the budget for the advertising campaign is set to a fixed value. Our choices correspond to the different combinations of budget allocation for the three sub-campaigns. Therefore, each time we are not pulling only one arm, but three, constraining the sum of the allocation for each sub-campaign to be equal to the cumulative budget. A bandit framework where a set of arms respecting some constraints instead of a single arm is pulled is said combinatorial [5].
We design a combinatorial bandit algorithm as follows, combining the three sub-campaigns. For each sub-campaign we create and initialize a specific regressor to understand and estimate the performance of the sub-campaign [7]. We consider and compare two regressors:

- Gaussian Process (GP) regressor
- Gaussian regressor

Then, for each day, we perform the following:

1. Find the best allocation of budgets by optimizing the combinatorial problem of the campaign (see details in Section 2.2.1);
2. Observe the reward from setting such allocation. Rewards are given by the amount of users clicks on our ads, w.r.t. Scenario 1.2.1 represented on Figure 1 (right);
3. Update the combinatorial bandit with the pulled arms and the observed rewards, respectively representing the allocated budget and the observed users visits for each sub-campaign.
4. Train each sub-campaign regressor.

2.2.1 Campaign Budget Optimization

Given an advertising campaign, defined by several sub-campaigns and a cumulative budget, we intend to find the best allocation of the budget over the sub-campaigns.
We do this resorting to a *dynamic programming* algorithm - taking inspiration from the classical knapsack problem [9] - which was presented during the course and whose formal model is hereby presented, without considering bidding for each advertisement.

At each timestep t:

$$\max_{y_{j,t}} \sum_{j=1}^{N} v_j n_j(y_{j,t})$$
$$\text{s.t.} \sum_{j=1}^{N} y_{j,t} \leq \bar{y}$$
$$y_{j,t} \geq 0 \quad \forall j \in [1, N]$$

where:

- $\{1, ..., N\}$ is the set of N sub-campaigns;
- $\bar{y}$ is the maximum cumulative budget;
- $y_{j,t}$ is the budget allocated to sub-campaign j at time t;
- v_j is the value per click of sub-campaign j (in this case $v_j = 1 \ \forall j$);
- n_j is the number of clicks of sub-campaign j, dependent on $y_{j,t}$.

Note that we are not considering bidding, more precisely, we assume bids are automatically optimized.

2.3 Results

Scenario 1.2.1 We run a simulation with the following parameters:

- Scenario: linear price and linear clicks (see Section 1.2.1).
- Number of arms for each sub-campaign bandit[1]: 11;
- Number of experiments: 100;
- Time horizon: 100 days;

We obtain the following results.

The implemented algorithm converges to the following budget allocation:

$$C_1 = 500\,€, \quad C_2 = 200\,€, \quad C_3 = 300\,€.$$

Observing the plot on the right in Figure 1, the proposed budget allocation intuitively seems the best.

[1]The number of arms corresponds to the number of evenly spaced discrete values of the cumulative budget.

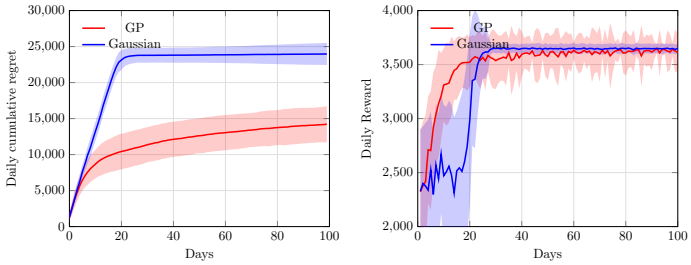

Figure 4: (left) Average daily cumulative regret with standard deviation intervals for scenario 1.2.1: (right) Average daily reward with standard deviation intervals for scenario 1.2.1

Regret at time t is defined as the difference between the reward obtained from the best budget allocation and the one from the t-th budget allocation. In a regret minimization framework, the instantaneous regret should tend to 0.
With reference to Figure 4, on the left we show the cumulative regret saturates over time, both with a GP and a Guassian regressor, proving the algorithm is indeed minimizing instantaneous regret. However, the bandit exploiting GP regressors accumulates less regret over time, showing better performances. This is because, as previously mentioned, the arms are correlated and by construction the GP regressor exploits this property [7].
Finally, on the right, the instantaneous reward - that is, the number of daily visits - grows very quickly over the first 20 days in the GP regressor case. Instead, it takes more time for the Gaussian regressor bandit to achieve higher reward. It then settles for both around 3600 visits per day.

3 Non-stationary advertising

3.1 Problem description

In this section, the total number of user clicks is maximized through advertisement algorithms that deals with non-stationary distributions. As already discussed in the introduction, we will deal with abrupt changes, and, in particular, three phases are considered.

3.2 Algorithm design choices

To deal with abrupt changes, we consider two approaches: sliding windows and change-point detection.

3.2.1 Sliding windows

Generally speaking, sliding approaches consists in using only data that has been collected recently to build estimates, thus allowing to capture changes in the distributions. These families of algorithms are simple and effective, and requires to set only one hyperparameter, that is the size sw of the sliding window (i.e. how long the agent should retain collected experience). We should note that if we choose a value for sw that is too small, we incur in bad performances, since the algorithm would seek for more exploration during its run, since its estimates will be inaccurate. Instead, if we fix a value for sw that is too large, we run into bad performances, since, at time t, the agent might estimate values with data that comes from a different distribution w.r.t the current one.

In the advertising problem that we are considering, and whose aim is to generate the maximum traffic to the website, the sliding window approach has been integrated with the algorithm discussed in Section 2 in the following way: the combinatorial problem is solved at each day t exploiting models (e.g. gaussian processes regressors) that are built using the rewards observed during the last sw days. The rest of the procedure follows inalterated.

3.2.2 Change-point detection

Algorithm 1 Change-Point Detection: CD$(w, b, Y_1, ..., Y_w)$

Require: an even number w indicating the window-size, w observations $Y_1, ..., Y_w$ and a prescribed threshold $b > 0$.

1: **if** $|\sum_{i=\frac{w}{2}+1}^{w} Y_i - \sum_{i=1}^{\frac{w}{2}} Y_i| > b$ **then**
2: Return True
3: **else**
4: Return False
5: **end if**

Change-point detection approaches, instead, are a families of algorithms that aim at detecting when a change in the distribution has happened: when such detection is triggered, an option is to reset the values of all the arms, and start learning from scratch. An example of such a mechanism is presented in Algorithm 1, taken from [4]: this simple component operates comparing running sample means over a sliding window. Note that the hyperparameter b is really important, it basically describes how easily we let the algorithm detects abrupt-changes. Therefore, finding a

good value for this hyperparameter is very important. Note that, fixing a value to b that is too large, we end up hurting the performance of the algorithm, since it will never detect a change in the probability distribution of the observations. On the other hand, fixing a value to b that is too small, we also end up hurting the performances, since a new phase will continuously be detected, thus starting again the exploration even though, the majority of the time, the optimal arm is unchanged.
Exploration is a central component in a change point-detection bandit, since without any expedient, arms believed to be sub-optimal are infrequently pulled: this might lead to fail in detecting abrupt changes in "infrequently visited" arms. In other words, we might have problems when the optimal arm in the previous phase do not change its distribution, while sub-optimal arms do. In order to solve this problem, one can resort to pull arms at random with a given probability γ.

In our particular advertising problem, we integrate the change-point detection with the algorithm discussed in Section 2 in the following way: the combinatorial problem is solved at each day t exploiting models (e.g. gaussian processes regressors) that are built using the rewards observed since the last change-point detection. Moreover, every time a super-arm is pulled, for each of its arm, we check whether the distribution of rewards for that arm has changed by means of the presented change-point detection mechanism. It is important to say that, every time we detect a change in any of the 3 sub-campaigns, we reset data for each of them. This is reasonable because, if a new phase is detected, it is very likely that the conversion rate probabilities change for all classes of users. Finally, as already discussed, superarms are pulled at random with probability γ, that is an hyperparameter. The rest of the algorithm remains the same.

3.3 Results

Scenario 1.2.3 We now present the results concerning the experiments of Scenario 1.2.3. We consider a total cumulative daily budget, as usual, of 1000 €, which has to be split among the three different sub-campaigns. As already discussed in the introduction, we deal with three phases, and each of them has optimal budget allocations. In the first phase, the best budget allocation corresponds to $[500, 200, 300]$, for C_1, C_2, C_3 respectively, with a maximum number of clicks of 3650. In the second phase, the best budget allocation is $[200, 800, 0]$, and the maximum number of clicks is 4600. In the third phase, the best budget allocation is $[0, 500, 500]$, and its maximum number of clicks is 4550.
Note that the first phase has a duration of 80 days, while the second one lasts 100 days and the third phase proceeds indefinitely. Therefore we decided to run our algorithms for 250 days, so that every phase is fairly included in the duration of the experiments.

We first compare the results in terms of daily visits, then we do the same in terms of cumulative regret. As you will see, it is particularly interesting to look at the results from the point of view of the the daily visits.
Daily Visits
Take into consideration the left graph shown in Figure 5: as the windows size w increases, the algorithm becomes more robust to changes in the observation distributions. As you can see, GPSW($w = 80$) outperforms the bandits with lower w values. This is due to the reason just explained above and due to the fact that the value of the window size is really similar to the duration of the phases.
Now take into consideration the right graph in Figure 5: note that the threshold b is really small, in practice this means that whenever an arm is pulled w times, we reset all the arms' values and

we start learning from scratch. This is exactly the reason why the CDBandit with parameter $w = 80$ outperforms the other one. Indeed, having $w = 80$ means that we reset the arms' values every 80 days, which is more or less the duration of the phases. On the other hand, with $w = 10$, we reset the arms every 10 days, leading to bad performances.

Also the γ value leads to significantly different performances. As you can see in Figure 6: the CDBandit having lower γ, precisely $\gamma = 0.05$, achieves better performances. This is due to the fact that the exploration is performed with lower probability, leading to a greater expected reward.

Now, we compare results given by CDBandits with different b values, shown in Figure 7. It is easy to see that the bandit with $b = 400$ outperforms the others. Remember that b is the threshold over which we detect a change in the distribution of the observations. As you can see, increasing b increases the stability of the algorithm. Indeed, having a high b value means that it is less likely that a change point is detected.

In the left graph shown in Figure 8 we compare the best CDBandits. It is easy to see that CD($w = 10, b = 400, \gamma = 0.1$) has the best performance.
In the right graph instead, we compare the best non-stationary Bandits with the GPBandit, which is designed to deal with stationary scenarios. As you can see, GPBandit has a "slower reaction" to phase changes. Indeed, this algorithm takes into consideration all the observations from day 0 to day $t - 1$, in order to decide which arm to pull at day t.At last i, we can see that GPSW generally achieves better expected performances w.r.t. CD, however the latter has a faster reaction to phase changes w.r.t. the former.
Cumulative Regret
Take into consideration the graph in the Figure 9: we can see that the best performance is achieved by GPSW($w = 80$). As expected, the highest cumulative regret is reached by the GPBandit. However, for most of the days, the worst performance is obtained by CD($w = 10, b = 0.1, \gamma = 0.1$), showing that, if poorly parameterized, the CD algorithm can lead to very poor performances. On the other hand, with good parameters, it can reach similar results to the ones of GPSW.

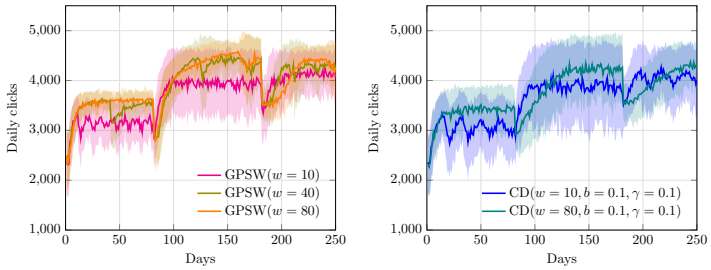

Figure 5: (left) Daily number of clicks, under scenario 1.2.3, obtained by GPSWBandit as a function of the sliding widow size. (right) Daily number of clicks, under scenario 1.2.3, obtained by CDBandit as a function of the sliding widow size.

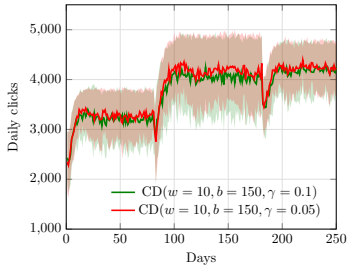

Figure 6: (left) Daily number of clicks, under scenario 1.2.3, obtained by CDBandit as a function of γ.

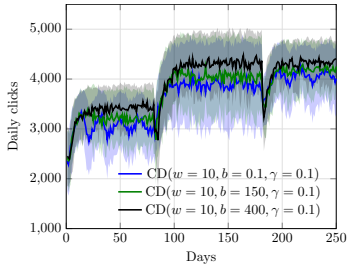

Figure 7: Daily number of clicks, under scenario 1.2.3, obtained by CDBandit as a function of b.

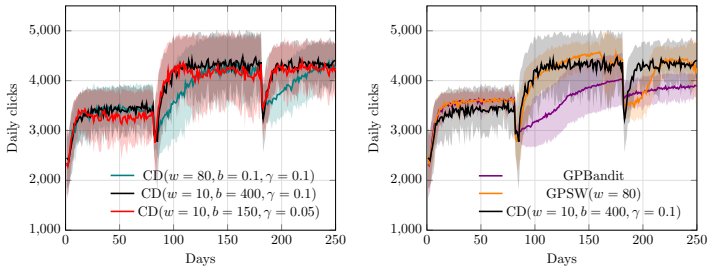

Figure 8: (left) Daily number of clicks, under scenario 1.2.3, obtained by CDBandits with the best performances. (right) Daily number of clicks, under scenario 1.2.3, obtained by the stationary GPBandit, and by the best non-stationary Bandits.

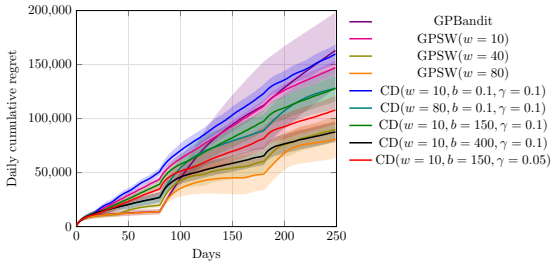

Figure 9: Daily cumulative regret, under scenario 1.2.3, obtained by the stationary GPBandit and by the non-stationary GPWSBandits and CDBandits.

4 Pricing under a fixed budget advertising scenario

4.1 Problem description

Here, the goal is to design a pricing algorithm under a scenario that satisfies the following assumptions:

- while advertising, a fixed daily budget is adopted for each sub-campaign;
- the process is stationary
- no user context is used for pricing the different users; thus the aggregated CRP is learnt and exploited by the learner

In particular, experiments are carried out in the following way: the advertising process is simulated using a fixed cost for each sub-campaign, thus generating traffic to the web-site: for each users that clicked on the ads, a learning algorithm decides a price ignoring the user features, and an observation for that price (i.e. bought or not bought) is obtained. User arrivals are simulated until a certain number of total users is reached.

4.2 Algorithm design choices

In order to perform the pricing task, we consider multiple bandit algorithms: UCB1 [2], TS [1], EXP3 [3], UCB1-M, UCB-L and UCB-LM. The last three algorithms have been analyzed in [8]: we believe them to be interesting since they are designed directly for pricing applications.
We now present some details about the algorithms adopted.

UCB1
The standard bandit algorithm, designed for Bernoulli variables is applied with a slight modification to handle the fact that observed rewards are not purely bernoullian. Indeed, at each round, the learning process observes a reward of $a_{i_t} x_{i_t}$, where i_t is the index of the pulled arm at round t, a_{i_t} is the value of the profit for such arm, and x_{i_t} is the realization of a Bernoulli random variable modelling the CRP. To handle such problem, the upper confident bound is calculated solely using the Bernoulli realizations, and, during arm selections, these bounds are multiplied by the known profit of the arm.

TS
Similar to UCB1, we adapted the TS algorithm to deal with the already mentioned problem: beta distributions are still fed with bernoullian realizations, and, during arm selections, samples from the beta distributions are multiplied by the known profit of the arm.

EXP3
EXP3 is based on importance-weighted sampling in order to balance the exploration/exploitation trade-off. A weight is kept for each arm in order to decide which one to pull next. Moreover, to perform exploration, the hyperparameter $\gamma \in [0, 1]$ is used, tuning the extent to which the arms are pulled uniformly at random. It is important to note that, as requested by the algorithm, at each round, the observed profit is scaled in the range $[0, 1]$ based upon the maximum possible profit.

UCB1-M
This bandit algorithm exploits the monotonicity property of the arms. This property can be assumed to hold in our case because we are performing pricing for a common good, not for a luxury

one. Thus, the cross rate probability is assumed to decrease as the price increases, satisfying the aforementioned property. This allows to bound the expected conversion rate of a given arm by taking into consideration also the experience collected by arms with higher conversion rates. In this way we may find a tighter bound.

UCB-L
This bandit algorithm exploits the low conversion rate property. When it is a priori known that the conversion rates are upper bounded by a value $\mu_{max} \leq \frac{1}{4}$, one can build probabilistic bounds that achieve better results than the one based on the Hoeffding's inequality.

UCB-LM
It combines the two previous techniques. Like in UCB1-M, the upper bounds of each arm are computed exploiting also the experience collected by arms with higher conversion rates. Then, differently from UCB1-M, the upper confidence bounds are built exploiting the Chernoff's inequality, instead of Hoeffding's one, and thus making use of the assumption over the low conversion rate probabilities, as it happens in UCB-L.

4.3 Results

Scenario 1.2.1 We now present the results concerning the experiments of Scenario 1.2.1. We consider a total cumulative daily budget of 1000 €, which we split equally among the 3 subcampaigns. In Figure 10 we shows the aggregated CRP and the expected profit curve under the advertising setting described above. The curves are obtained by a weighted average of the respective user-context curves, where the weights are given by the percentage of users of each class in a day, considering the fixed advertising strategy. From this curves, it is possible to obtain the clairvoyant algorithm, providing a best expected profit of 0.5775 € reached for a price of 17.55 €. In a daily expectation this corresponds to 1808.83 €. Considering a uniform discretization of the interval $[15, 25]$ in 11 arms, we obtain that the best price is 18 €, leading to 0.573 € expected profit for each day, and a daily expected profit of 1797.12 €.
Daily cumulative regret and daily expected profit are shown in Figure 11. We should note that the regret has been calculated with the use of the discrete best price; the plot of the regret with the continuous optimal price has not been reported since it is very similar, due to the high similarity between these optimal values. However, considering the true optimal price, we have to expect to suffer a linear regret, since, at each round, some profit is lost due to price discretization. As hyperparameters, γ for EXP3 is set to 0.1, and the upper bound on μ_{max} is set to 0.3 for UCB-L and UCB-LM.

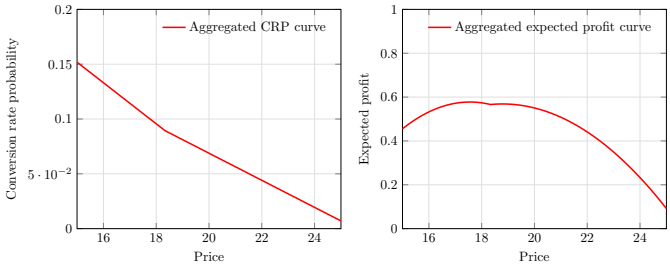

Figure 10: (left) Aggregated CRP under scenario 1.2.1 and a fixed advertising strategy. (right) Aggregated expected profit under scenario 1.2.1 and a fixed advertising strategy

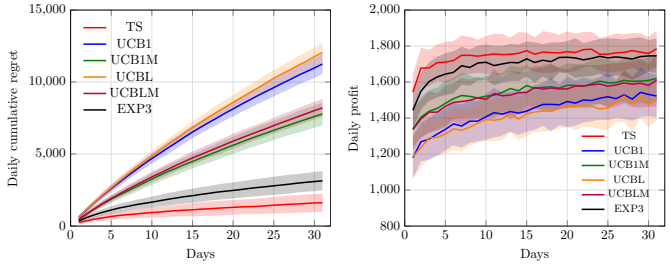

Figure 11: (left) Average daily cumulative regret with standard deviation intervals for scenario. 1.2.1: (right) Average daily profit with standard deviation intervals for scenario. 1.2.1

Scenario 1.2.2 Results concerning the experiments on Scenario 1.2.2 are now discussed. Again, we consider a total cumulative daily budget of 1000 €, equally divided among the 3 sub-campaigns. Aggregated CRP and expected profit curves are calculated in the same way discussed for the previous scenario. Plots are shown in Figure 12. In this case, we obtain that the optimal price would be 17.44, obtaining 0.5276 € and 1652.72 € expected profit per user and daily, respectively. Considering, instead, uniform discretization of the interval of interest in 11 possible prices, we obtain 17 as best price, with a user profit of 0.52 €, and a daily profit of 1630.66 €.
Daily cumulative regret and expected profit are shown in Figure 13. Again, the same reasoning concerning discrete and continuous optimal point holds also here.
As hyperparameters, γ for EXP3 is set to 0.1, and the upper bound on μ_{max} is set to 0.3 for UCB-L and UCB-LM.

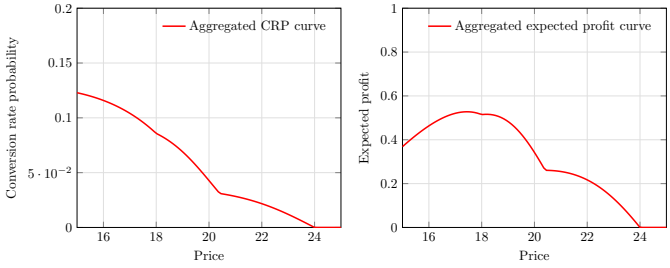

Figure 12: (left) Aggregated CRP under scenario 1.2.2 and a fixed advertising strategy. (right) Aggregated expected profit under scenario 1.2.2 and a fixed advertising strategy

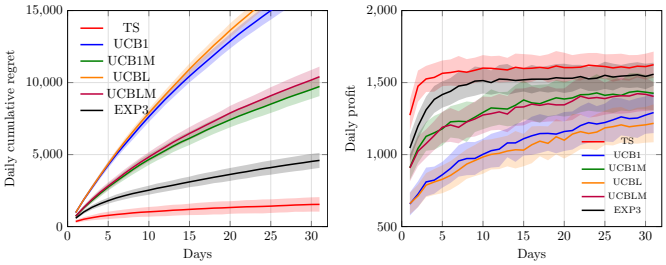

Figure 13: (left) Average daily cumulative regret with standard deviation intervals for scenario. 1.2.2: (right) Average daily profit with standard deviation intervals for scenario. 1.2.2

5 Contextual pricing under a fixed budget advertising scenario

5.1 Problem description

Differently from the previous scenario, here, the goal is to discriminate the users in different contexts in order to improve the profit under the following assumptions:

- the budget allocation to each subcampaign is fixed;
- unique phase;
- weekly context generation.

Indeed, this problem is very similar to the one of Section 4 but with the exception of the context generation. Therefore, we need to design a similar algorithm with the ability to generate contexts in order to discriminate users based on their features.

5.2 Algorithm design choices

The general algorithm to solve this problem is straightforward. For every week:

1. Run the context generation algorithm to build a context structure
2. Construct a bandit for each context and update its parameters by using the old collected data (notice that the bandit can be any of the ones developed in Section 4)
3. For each user that visits the website, use the corresponding bandit (i.e. the one matching the user's features) to choose the price

Even in this case, there are design choices to be made on the context generation algorithm. Among the different types, we choose to design a *greedy* algorithm due to its small complexity and exhaustive capacity. More in details, the problem of context generation is a very hard one due to the exponential number of partitions; thus, since the number of features we are considering is just 2, we also implemented a *bruteforce* algorithm

5.2.1 Greedy context generation algorithm

The greedy algorithm developed is exactly the one explained in the course of DIA, but for clarity, it is also explained here. Basically, the goal of the algorithm is to build a binary tree where each edge represents the feature that has been splitted. Then, each path from the root to a leaf corresponds to a context belonging to the generated context structure. In order to generate the tree, at each node, the following algorithm is run:

1. For every feature i, evaluate the value v_i after the split of context c_0 into (c_1, c_2)

   $$v_i = \underline{p}_{c_1} \underline{\mu}_{a^*_{c_1}, c_1} + \underline{p}_{c_2} \underline{\mu}_{a^*_{c_2}, c_2}$$

   where:

   - c_0 is the current context considered at the node
   - (c_1, c_2) are the two contexts obtained after the split
   - $\underline{p}_{c_i}$ is the lower bound on the probability that context c_i occurs

- $\underline{\mu}_{a^*_{c_i},c_i}$ is the lower bound on the best expected reward for context c_i
- $a^*_{c_i}$ is the optimal arms for context c_i

2. Among all the values

$$v_i \geq \underline{\mu}_{a^*_{c_0},c_0}$$

where $\underline{\mu}_{a^*_{c_0},c_0}$ is the lower bound on the best expected reward for context c_0, select the feature with the largest value and split it

Regarding the lower bounds previously defined, we choose to use a variation of the *Hoeffding* bound since the reward variable is a Bernoulli distribution multiplied to the profit; thus, it can be normalized by scaling the value of the bound:

$$HoeffdingLowerBound = \bar{x} - s\sqrt{-\frac{\log \delta}{|x|}}$$

where:

- x is the realizations of a constant-scaled Bernoulli random variable X
- δ is the confidence in terms of probability of error (small value)
- s is the scale value which is the maximum of the realizations

Moreover, the optimal arm for the best expected rewards is obtained by doing an offline training of the bandit using all the previous collected data.

5.2.2 Bruteforce context generation algorithm

The bruteforce algorithm simply compares all the possible partitions by calculating their value

$$v_i = \sum_{c_i} \underline{p}_{c_i} \underline{\mu}_{a^*_{c_i},c_i}$$

where given a partition, c_i is a context belonging that partition and all the other variables are the same defined in the Greedy algorithm. Then, the chosen context structure is the one with the highest value.

5.2.3 Further design choices

As the context generation algorithm is defined until now, there is a critical issue: in the beginning, at every week the algorithm generates a new context structure from zero by using all the data collected until the start, but this introduces a huge bias on the expected rewards of each arm. For clarity, we show an example:

- suppose to have the following assumptions:

  1. start the algorithm with a unique context aggregating for the first week
  2. during the start of the second week there is a split in a feature and the optimal arm is a different one w.r.t. first week that gives a larger profit

- under these assumptions, in the third week, the algorithm with high probability will choose to use the same context structure of the first week because the data tells that the optimal arm is the one of the second week

In order to solve this problem, we choose to use an *"incremental"* context generation, i.e. use the context generation algorithm on top of the previous context structure to avoid the re-aggregation of contexts.

5.3 Results

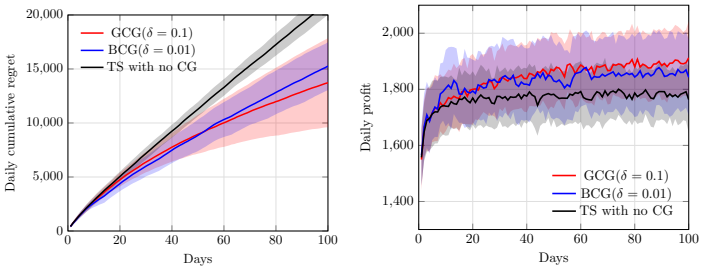

Figure 14: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1: (right) Average daily profit with standard deviation intervals for Scenario 1.2.1

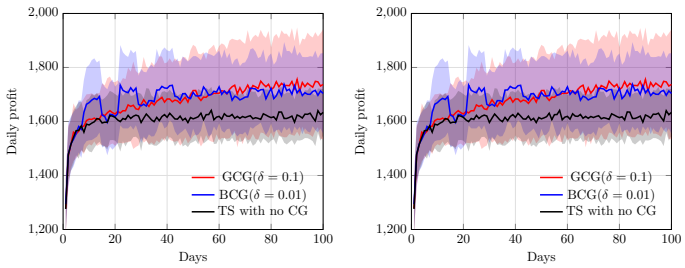

Figure 15: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2

Scenario 1.2.1 The results concerning the experiments of Scenario 1.2.1 are discussed here. For comparison reason, we consider a total cumulative daily budget of 1000 € split equally among the 3 sub-campaigns. In this case, the optimal price for each class of user is [18 €, 19 €, 15 €] by considering a uniform discretization of the interval in 11 possible prices. Which means that the optimal daily profit is 1979.06 €. The experiments are executed only for the Thompson Sampling bandit defined in the Section 4 since it is the one with highest performance and the results are shown in Figure 14. It shows some behavior as the confidence changes and the other parameters of each single bandit are the same of the Section 4.

Scenario 1.2.2 We present the results about the experiments of Scenario 1.2.2. Again, we consider a total cumulative daily budget of 1000 € split equally among the sub-campaigns. In this instance, if we consider a uniform discretization of 11 prices, the optimal price for each class of user is [20 €, 18 €, 16 €]. Thus, the optimal daily profit is 1982.02 €. The experiments are

executed for every bandit defined in the Section 4 and the results are shown in Figure 15. For the other parameters of each single bandit, they are the same of the Section 4.

6 Joint optimization of pricing and advertising

6.1 Problem description

The goal is to combine the two main problems of the previous chapters, namely the allocation of the budget and the pricing. Here, an important assumption is taken into consideration: it is a-priori known that every sub-campaign is associated with a different context, and that a different price is charged for each class of users.
In order to carry out this task, we optimize the budget allocation over the three sub-campaigns to maximize the number of clicks through a combinatorial bandit algorithm , and, at the same time, perform pricing taking into account that each user belongs to a class, thus user context is also taken into consideration by the learning algorithm.

6.2 Algorithm design choices

Due to the a-priori knowledge about the association of sub-campaign and class of user, the two problems of pricing and allocation of budget can be tackled independently and combined later with the computation of the value per click of each sub-campaign. Thus, the algorithm to deal with this problem is based on the combination of the algorithms explained in Section 2 and 4. Nonetheless, there are still design choices to be made regarding how to incorporate the value per click for each sub-campaign inside the algorithm. We show two ways to do it:

1. Learn directly the daily value of each sub-campaign by multiplying the number of clicks with the value per click;
2. Learn the daily number of clicks of each sub-campaign and only during the optimization problem of the advertising problem introduce the value per click.

Moreover, we estimate the value per click with two different methods:

1. *Expectation* of the rewards of each sub-campaign

   $$\hat{v}_j = \mu_{j,t-1}$$

   where:

   - $\mu_{j,t-1}$ is the expected rewards of sub-campaign j considering the data upon time $t-1$

2. *Quantile-based exploration* of the rewards of each sub-campaign (shown in Nuara et al. [6])

   $$\hat{v}_j = q\left(\mu_{j,t-1}, \sigma^2_{j,t-1}, 1 - \frac{1}{t}\right)$$

   where:

   - $q(\mu, \sigma^2, p)$ is the quantile of order p of a Gaussian distribution with mean μ and variance σ^2
   - $\sigma^2_{j,t-1}$ is the variance of the rewards of sub-campaign j considering data upon time $t-1$

For clarity, the general pseudocode is reported in Algorithm 2. For the different variation of the algorithm, we call it by adding a suffix -Q and -Exp for the calculation method of value per click; while we add a suffix -V for the algorithm that learns the number of clicks instead of the overall value of the sub-campaign.

6.2.1 Improved version

In the previous part, the value per click is calculated by considering all the rewards of a specific sub-campaign/class. With this improved version, the calculation is done only with the best rewards which are the rewards obtained by using the optimal arm estimated by the bandit itself. By applying this improvement, the agent is able to estimate better the correct value per click of the sub-campaign.
To differentiate this algorithm with the previous ones, we call it Joint Bandit Improved (JBI) followed by Quantile (QV) or Expectation (ExpV) indicating the method to calculate the value per click and the fact that it learns visits of users (i.e. number of clicks).

Algorithm 2 JB: Joint price and advertising optimization bandit with discrimination

Require: number of days n, list of possible prices to be used.

Initialize current budget allocation b_t at random.
Initialize current value per click $v_{j,t}$ of each sub-campaign at 1
for $t = 1, 2, \ldots, n$ **do**
  Spend budget b_t and observe m users visiting
  for $u = 1, 2, \ldots, m$ **do**
    Price the user u according to the bandit associated to the corresponding user features
    Update the bandit associated to the corresponding user features with collected reward
  end for
  Update regressor model with collected data (number of clicks or value of sub-campaign)
  Sample regressor model for each sub-campaign
  Update the value per click $v_{j,t}$ using an estimation method (expectation or quantile)
  Solve the optimization problem by using the value per click $v_{j,t}$ of each sub-campaign and fix b_{t+1} as the budget
end for

6.3 Results

Scenario 1.2.1 Here, we discuss the experiments of Scenario 1.2.1. Due to the huge amount of combinations of pricing bandits and advertising bandit, we run only experiments with Thompson sampling bandit algorithm and combinatorial bandits with gaussian processes. In this case, with the possibility to discriminate users, the conversion rate probabilities and number of clicks functions are the original ones, and the optimal budget allocation and prices of each class of user/sub-campaign is respectively [500€, 200€, 300€] and [18€, 19€, 15€] by considering a uniform discretization of the interval in 11 possible prices and 11 possible budgets. Overall, the optimal daily profit is 2327.18€. As hyperparameters of the pricing bandit and advertising bandit, we set them exactly like the previous experiments of Section 2 and 4. Then, we run the experiments for all combinations of learning type and value per click estimation methods and the results are shown in Figure 16. Notice that the improved version algorithm, JBIQV and JBIExpV, performs a little better than the others since it estimates the value per click more precisely.

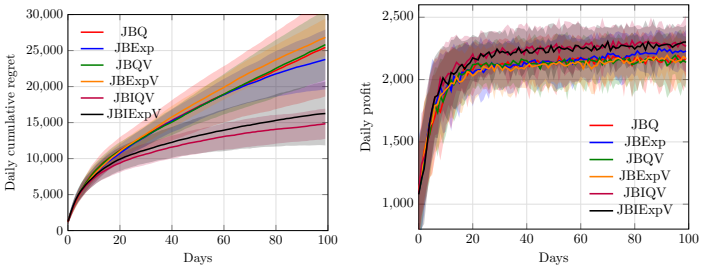

Figure 16: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1: (right) Average daily profit with standard deviation intervals for Scenario 1.2.1

Scenario 1.2.2 In this case, the experiments of Scenario 1.2.2 are discussed. Regarding the pricing and advertising bandit, like in the previous case, we choose to use only Thompson sampling and gaussian processes based combinatorial bandits. In this scenario, the optimal budget allocation for each sub-campaign is [500 €, 200 €, 300 €] considering 11 uniform discretized arms, while the optimal prices of each class of user is [20 €, 18 €, 16 €] with 11 arms like budget-case. Overall, we obtain a daily profit of 2365.44 €. Meanwhile, as hyperparameters, we set them exactly like the previous scenario. Moreover, the experiments are run for all combinations of learning type and value per click estimation methods; the results are shown in Figure 17.

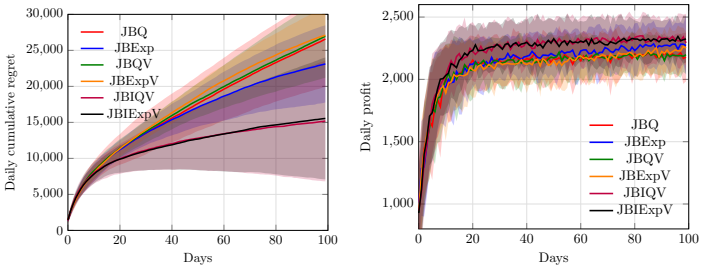

Figure 17: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2

7 Joint optimization of pricing and advertising without class information

7.1 Problem description

The problem here is the one of jointly optimizing pricing and advertising strategy under the constraint that the pricer should make the same price to all the classes of users. This assumptions can be interpreted in, at least, two ways:

- While users arrives the context can't be used to make the price, even if we can observe it. In such a case, we can still change the price for every user that arrives to the website: this is a simplification w.r.t. a real-case setting.
- Under a more realistic hypothesis, we are forced to make, for a whole day, the same price to all the users.

We should note that, disregarding the interpretation, the problem is, indeed, non-trivial: the joint optimization, under such constraint gets more complex, since the advertising and the pricing strategies interfere with each other much more w.r.t. the previous point, and the optimization becomes a joint problem. Indeed, if we could make a different price discriminating with the context, as in the previous point, the problem of pricing and advertising, as already seen, can be totally decoupled: such an approach, here, is no more possible.
To better clarify this point, consider the following: when the agent should decide the budget to be allocated for the next day, he should be able to estimate the number of clicks that each sub-campaign will generate, but also the click-value for each of the sub-campaign. In the case in which the agent can discriminate prices by means of user features, he can estimate, for each context, the click-value, depending on the best possible price that he can make to that context (as shown in the previous context). If, instead, the agent is deprived from this possibility, the click-value must take into account the distribution that he will face in terms of user context in the next day, otherwise, he may incur in sub-optimal strategies. Indeed, a given price could be optimal for certain users, while being highly sub-optimal for others: if the agent makes that price without investing budget in the right sub-campaign, the agent will face a high regret: in this sense, the problem of pricing and advertising can't be decoupled, but it must be faced jointly.

7.2 Algorithm design choices

Algorithms for facing the previous problem are here discussed. We first present an algorithm that deals with the assumptions that prices can be made user-per-user without considering the context, and, then, an algorithm to deal with the more realistic assumption is presented.

7.2.1 Joint bandit with balanced distribution of data

As discussed before, the problem of pricing and advertising has to be considered at the same time. In order to deal with the advertising side problem, an idea is to consider both problems by solving as many optimization problems as the number of pricing arms. Each problem uses the same sampled number of clicks, but a different value per click estimated from the rewards of the arm considered. Among the different solutions of the problems, the best budget allocation chosen is the one that gives the largest overall value, i.e. sum of number of clicks multiplied by the value per click. By doing this, the budget allocation problem is handled correctly by considering also the pricing problem.

Regarding the pricing problem, there is still an issue about the variability in the daily number of clicks. Basically, each day the distribution of users visiting depends on the advertising problem, and this can be seen as a multiple phases situation. Indeed, this issue can be dealt with more advanced techniques used in Section 3 such as an incremental sliding window. But, since there is already a model estimating the distribution of users, it can be exploited to solve this problem. Essentially, the solution is to to balance the distribution of data for each arm of the unique pricing bandit in order to be as close as possible to the distribution of users during the current day. This might be bad during the beginning due to the bad estimation of number of clicks, but it will gradually get better as the estimation becomes more precise.

Overall, the core of the algorithm is based on the two previous ideas:

1. Select the best budget allocation by solving n_p optimization problems with different value per click where n_p is the number of arms of the pricing problem
2. Each day, balance the distribution of data used in the unique pricing bandit by maintaining it as close as possible to the estimated distribution of users

Regarding the regressor model of the number of clicks and the unique pricing bandit, any of the ones developed in Section 2 and 4 can be used. Moreover, the value per click is estimated by using the same method shown in 6.1. For a more detailed structure of the algorithm, the pseudocode is shown in the Algorithm 3 that is referred with the name of JBB (to differentiate the methods about the value per click estimation we use a suffix -Exp and -Q). For comparison, there are also algorithms with suffix -D that indicates fixed daily price.

7.2.2 Joint bandit with fixed price for a whole day

In this situation a fixed price is made for all users that will visit the web-site during the next day. In order to carry this out, we consider the following approach. Let's start with the estimators that are used to solve the problem: first of all, considering the number of clicks per budget spent in each sub-campaign, gaussian processes regressors are used in the same way of Section 2. At each day, the agent observes the distribution of user contexts that arrives on the web-site, and, once this information is collected, a new data-point for the gaussian process is available to refine the model. Moving the focus to the pricing aspect of the problem, the agent, while users arrive on the website, will collect information, for each context, about the CRP of the different users at the value determined by the fixed price chosen at the previous day. In particular, the success of purchase is used to update beta distributions (one for each context) in a Thompson sampling manner.
Once all the estimates of a given day are collected, the agent needs to solve the campaign optimization problem, trading-off exploration and exploitation, to decide the price and the budget to be used for the next day. To solve this issue samples, the agent exploits the gaussian processes and the beta distributions: number of clicks for each sub-campaign, and profit for each price and sub-campaign, are obtained by sampling the gaussian processes and the beta distributions respectively. Once this sampling phase is done, the agent uses these estimates and iterates among the possible prices, solving, with dynamic programming, the budget allocation process: the price that achieves the larger expected profit among the possible is the one that will be used for the next day, together with the respective budget that gives raise to such revenue. The pseudo-code is reported in Algorithm 4: we will refer to it with the name of JBFTS (joint bandit fixed thompson sampling). We should note that, in principle, other explorations strategies are possible for pricing, such as upper confidence bounds.

Algorithm 3 JBB: Joint bandit with balanced distribution of data

Require: number of days n, list of possible prices to be used.

Initialize current budget b_t at random.
Initialize empty unique pricing bandit ban_t
for $t = 1, 2, \ldots, n$ **do**
  Spend budget b_t and observe m users visiting
  for $u = 1, 2, \ldots, m$ **do**
    Price the user u by pulling the bandit ban_t
    Save the observed reward globally and update the bandit ban_t
  end for
  Update the regression model with collected data
  Sample the regression model for each sub-campaign
  Update value per click estimation for each price
  Solve the optimization problem for each price and select the new budget b_{t+1} that maximize overall value of the campaign
  Initialize a new unique pricing bandit ban_{t+1} and re-train it in an offline fashion by using a balanced number of data for each arm
end for

Algorithm 4 JBFTS: Joint price and advertising optimization under fixed price constraint for a whole day

Require: number of days n, list of possible prices to be used.

Initialize current price p_t and current budget b_t at random.
for $t = 1, 2, \ldots, n$ **do**
  Spend budget b_t and price users who click on the advertising with p_t while updating the beta distributions of the observed user context and user price
  Update Gaussian Process model with collected data
  Sample Gaussian Process for each sub-campaign
  Sample beta distributions for each price and each sub-campaign
  Solve the optimization problem for each price, exploiting the values sampled, and fix p_{t+1} and b_{t+1} as the prices and the budget that achieves the best expected profit
end for

7.3 Results

Scenario 1.2.1 We now present the results concerning the experiments of Scenario 1.2.1. We consider a total cumulative daily budget, as usual, of 1000 €, which has to be split among the three different sub-campaigns. In this situation, the best budget allocation corresponds to $[500, 200, 300]$, for C_1, C_2, C_3 respectively. The optimal price is 18 €, providing an expected daily profit of 2161.08 €. We run our algorithms for 50 days; results are shown in Figure 18. When D is appended to the name of already presented bandit, it means that the mechanics of the bandit remain the explained ones, however, a price is fixed for the whole day. We note that JBBQ and JBBExp reach higher performances due to the fact that they are not constraint to use the same price for a whole day: indeed, they can optimize prices online user by user.

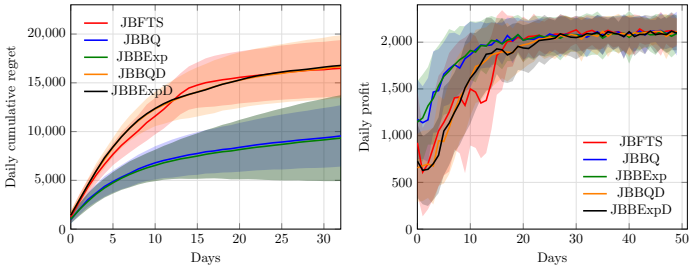

Figure 18: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1: (right) Average daily profit with standard deviation intervals for Scenario 1.2.1

Scenario 1.2.2 For what concerns Scenario 1.2.2, we consider, again a total cumulative daily budget of 1000 €. The best budget allocation is, differently from the previous point, $[500, 200, 0]$, and the best price is 18 €: indeed, with such price, C_3 retrieves no profit (the CRP is 0). The daily expected revenue results in 1949.78 €. Figure 19 shows results for 50 days.

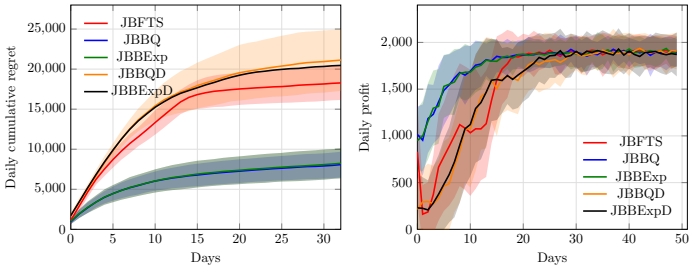

Figure 19: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2

A Additional Experiments

A.1 Additional Scenarios

A.1.1 Linear Scenarios

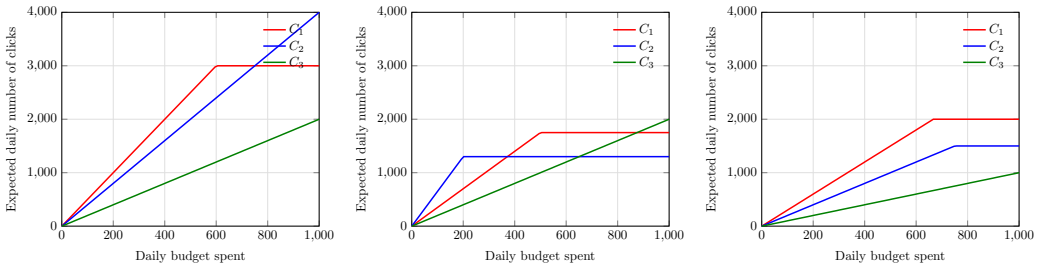

Figure 20: Expected daily number of visits per daily budget spent and user context. (left) Different linear coefficients and upper bounds. (middle) Different noise standard deviation. (right) Different linear coefficients and upper bounds of function estimating number of clicks.

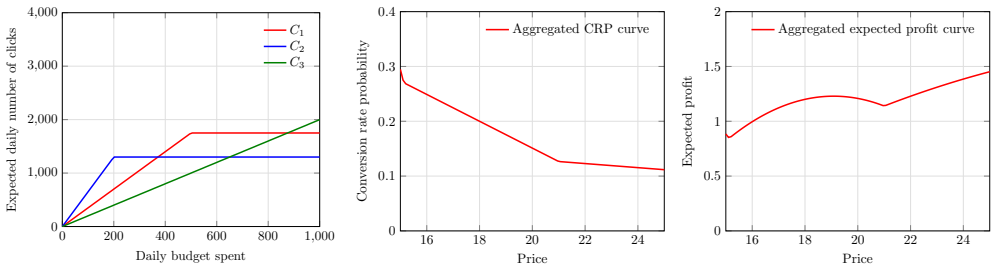

Figure 21: Expected daily number of visits per daily budget spent and user context. (left) Different CRP function. (middle) Aggregated CRP. (right) Aggregated expected profit.

A.1.2 Tanh Scenarios

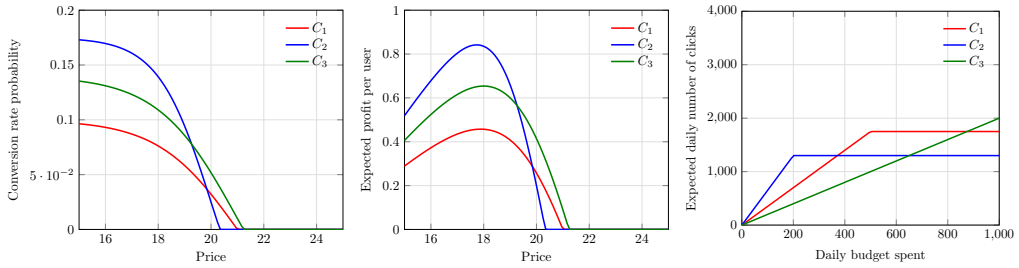

Figure 22: (left) Conversion rate probabilities per user context. (middle) Expected profit per user context. (right) Expected daily number of visit per daily budget spent and user context.

A.2 Stationary Advertising

A.2.1 Changing the number of arms

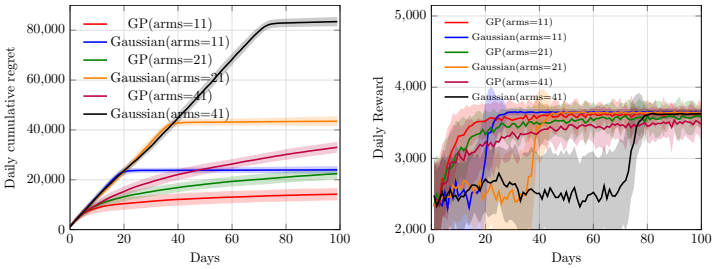

Figure 23: (left) Average daily cumulative regret with standard deviation intervals for scenario 1.2.1: (right) Average daily reward with standard deviation intervals for scenario 1.2.1

A.2.2 Changing Linear Scenario Parameters

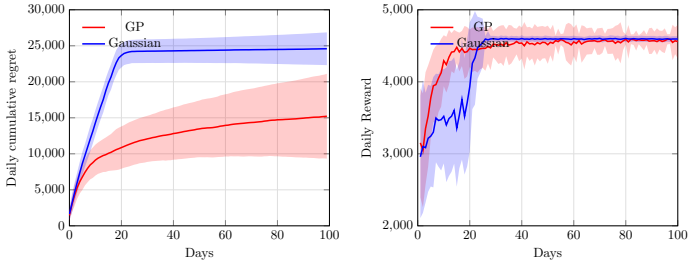

Figure 24: (left) Average daily cumulative regret with standard deviation intervals. (right) Average daily reward with standard deviation intervals. Scenario Fig. 20 left.

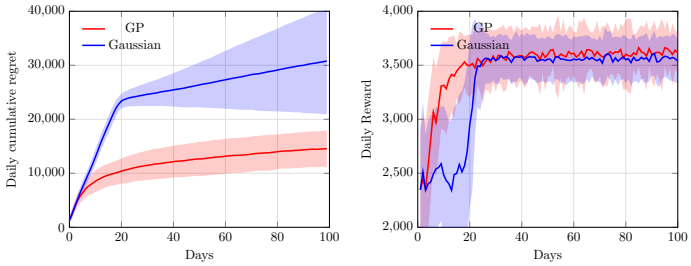

Figure 25: (left) Average daily cumulative regret with standard deviation intervals. (right) Average daily reward with standard deviation intervals. Scenario Fig. 20 middle.

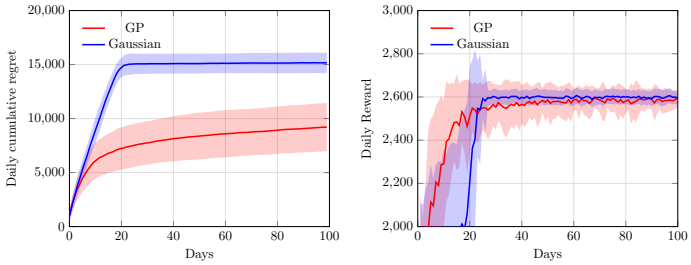

Figure 26: (left) Average daily cumulative regret with standard deviation intervals. (right) Average daily reward with standard deviation intervals. Scenario Fig. 20 right.

A.3 Non-Stationary Advertising

A.3.1 Changing Non-Stationary Scenario Phase Duration

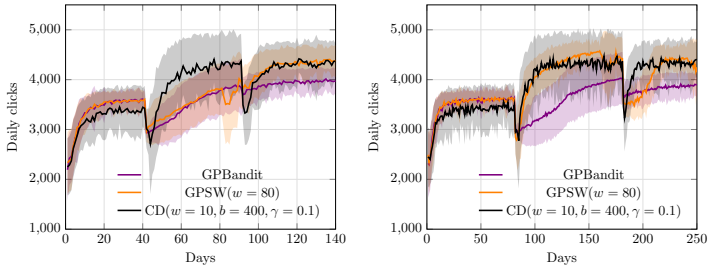

Figure 27: (left) Daily number of clicks under under scenario 1.2.3 with shorter phases. (right) Daily number of clicks under under scenario 1.2.3

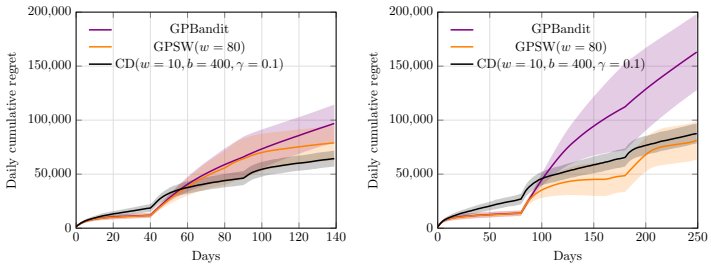

Figure 28: (left) Daily cumulative regret under scenario 1.2.3 with shorter phases. (right) Daily cumulative regret under scenario 1.2.3

A.3.2 Changing Non-Stationary Scenario STD

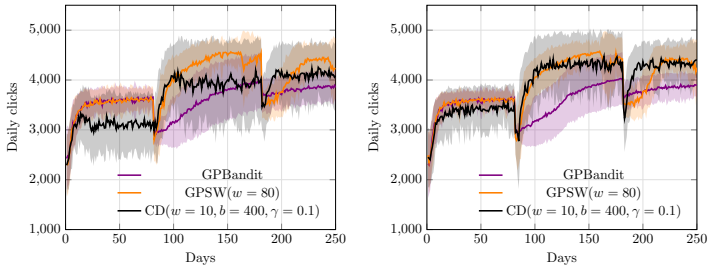

Figure 29: (left) Daily number of clicks under scenario 1.2.3 with $std = 100$. (right) Daily number of clicks under scenario 1.2.3 with default $std = 25$

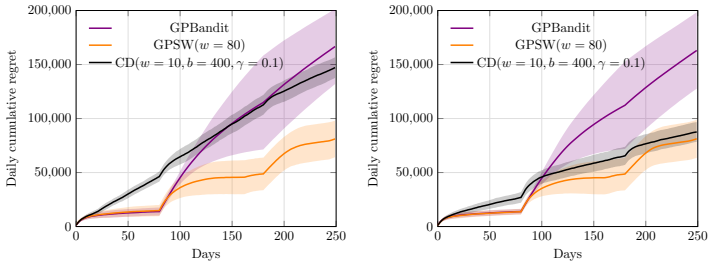

Figure 30: (left) Daily cumulative regret under scenario 1.2.3 with $std = 100$. (right) Daily cumulative regret under scenario 1.2.3 with default $std = 25$

A.4 Pricing under a fixed budget

A.4.1 Changing Linear Scenario CRP

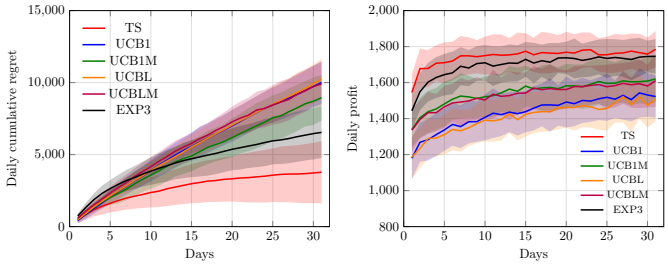

Figure 31: (left) Average daily cumulative regret with standard deviation intervals. (right) Average daily reward with standard deviation intervals. Scenario Fig. 21.

A.4.2 Changing Linear Scenario Clicks Function

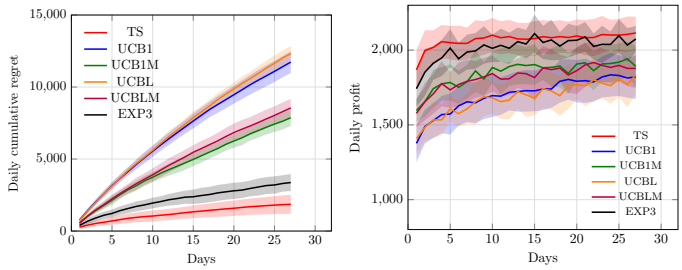

Figure 32: (left) Average daily cumulative regret with standard deviation intervals. (right) Average daily reward with standard deviation intervals. Scenario Fig. 20 left. Same CRP as 1.2.1.

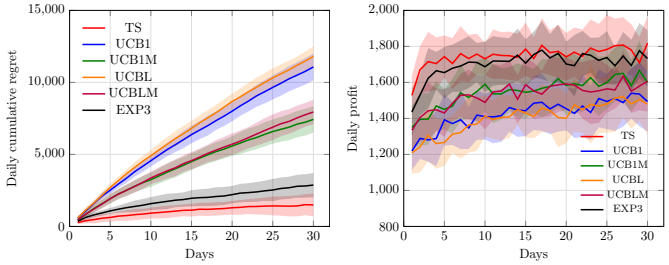

Figure 33: (left) Average daily cumulative regret with standard deviation intervals. (right) Average daily reward with standard deviation intervals. Scenario Fig. 20 middle. Same CRP as 1.2.1.

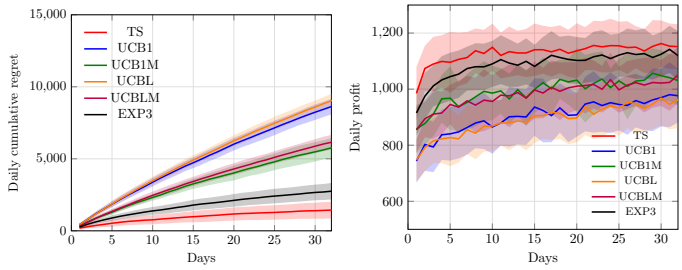

Figure 34: (left) Average daily cumulative regret with standard deviation intervals. (right) Average daily reward with standard deviation intervals. Scenario Fig. 20 right. Same CRP as 1.2.1.

A.4.3 Changing Tanh Scenario CRP

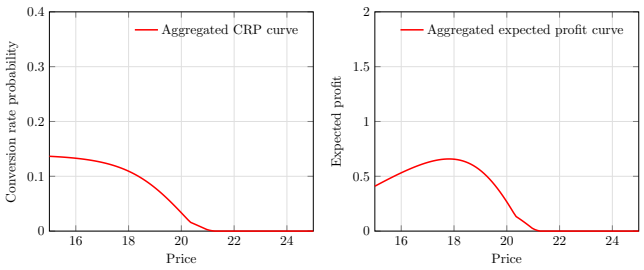

Figure 35: (left) Aggregated CRP under Scenario Fig. 22 and a fixed advertising strategy. (right) Aggregated expected profit under Scenario Fig. 22 and a fixed advertising strategy

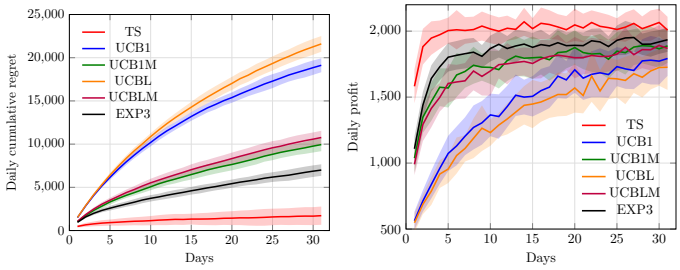

Figure 36: (left) Average daily cumulative regret with standard deviation intervals for Scenario Fig. 22. (right) Average daily profit with standard deviation intervals for Scenario Fig. 22

.

A.5 Contextual pricing under a fixed budget advertising scenario

A.5.1 Changing probability of error δ

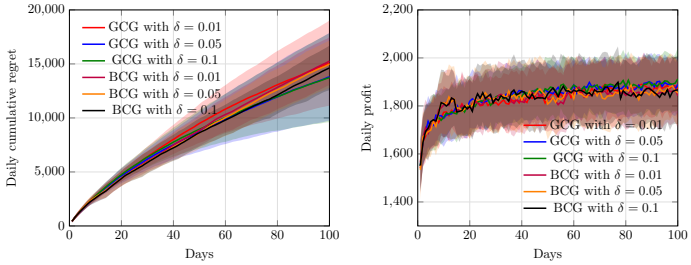

Figure 37: left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1: (right) Average daily profit with standard deviation intervals for Scenario 1.2.1; all done with 100 runs

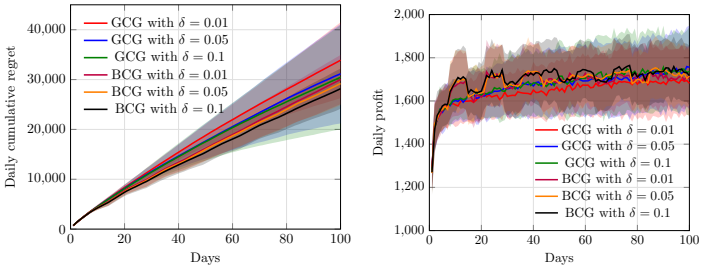

Figure 38: left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2; all done with 100 runs

A.5.2 Unique class scenario

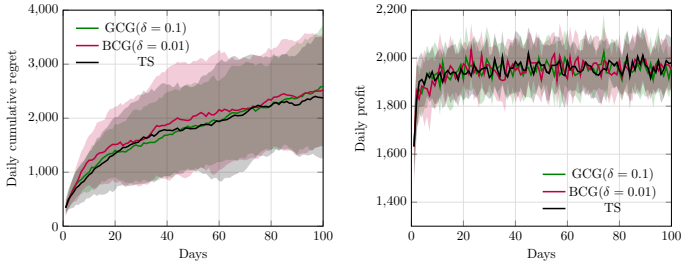

Figure 39: Unique class scenario to check if there is difference w.r.t. no-context generation algorithm; (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1 but all class equal C_1: (right) Average daily profit with standard deviation intervals for Scenario 1.2.1 but all class equal C_1; all done with 20 runs

A.5.3 Changing pricing arms

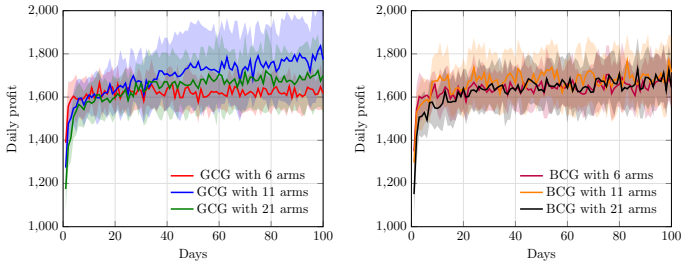

Figure 40: GCG done with $\delta = 0.1$, while BCG done with $\delta = 0.0001$; both graphs represents average daily cumulative regret with standard deviation intervals for Scenario 1.2.2; All done with 20 runs

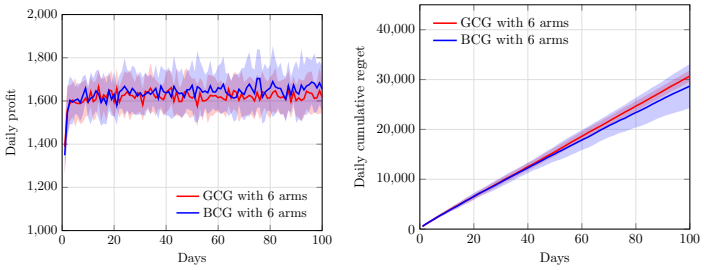

Figure 41: GCG done with $\delta = 0.1$, while BCG done with $\delta = 0.0001$; (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2; all done with 20 runs

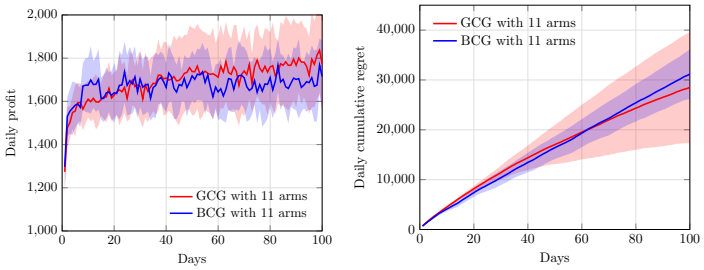

Figure 42: GCG done with $\delta = 0.1$, while BCG done with $\delta = 0.0001$; (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2; all done with 20 runs

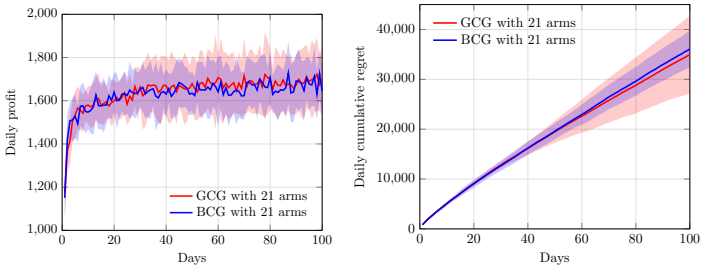

Figure 43: GCG done with $\delta = 0.1$, while BCG done with $\delta = 0.0001$; (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2; all done with 20 runs

A.5.4 Changing pricing bandit

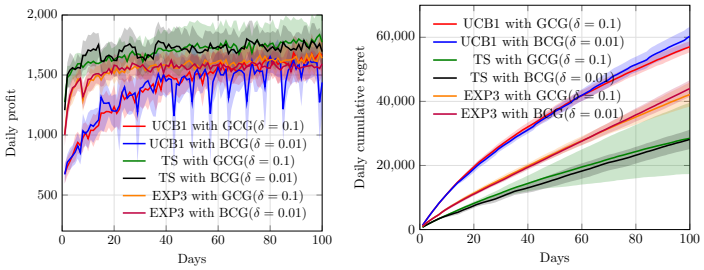

Figure 44: GCG done with $\delta = 0.1$, while BCG done with $\delta = 1e - 10$; (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2; all done with 20 runs

A.6 Joint optimization of pricing and advertising

A.6.1 Changing Tanh Scenario Standard Deviation

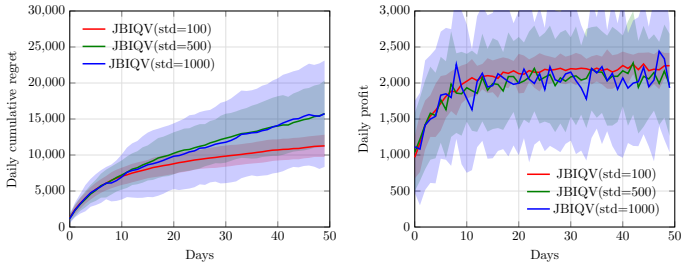

Figure 45: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2

A.6.2 Changing Linear Scenario Budget

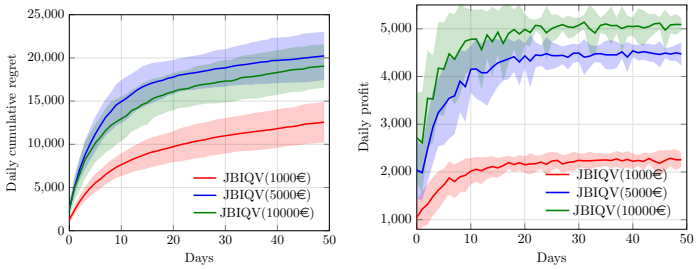

Figure 46: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1: (right) Average daily profit with standard deviation intervals for Scenario 1.2.1

A.6.3 Changing Pricing Bandit

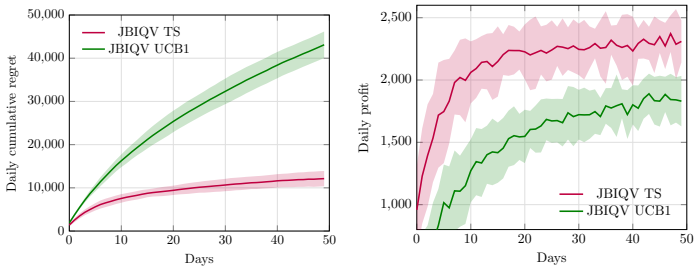

Figure 47: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.2: (right) Average daily profit with standard deviation intervals for Scenario 1.2.2

A.6.4 Changing Pricing Arms

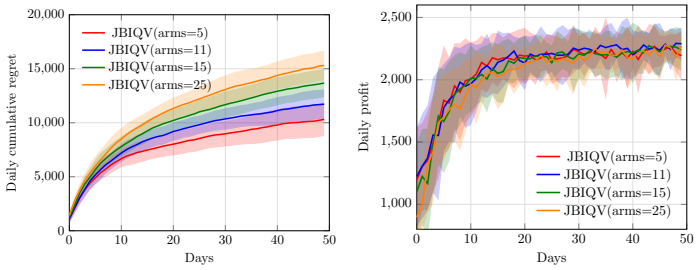

Figure 48: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1: (right) Average daily profit with standard deviation intervals for Scenario 1.2.1

A.6.5 Changing Advertising Arms

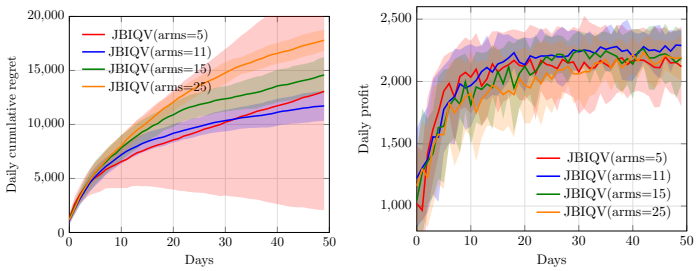

Figure 49: (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1: (right) Average daily profit with standard deviation intervals for Scenario 1.2.1

A.7 Joint optimization of pricing and advertising without class information

A.7.1 Changing standard deviation of number of clicks

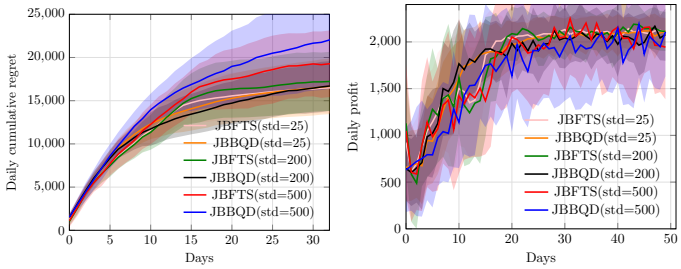

Figure 50: Arms=5. (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1. (right) Average daily profit with standard deviation intervals for Scenario 1.2.1. Last number in the legend indicates the standard deviation used to generate the data.

A.7.2 Changing cumulative daily budget

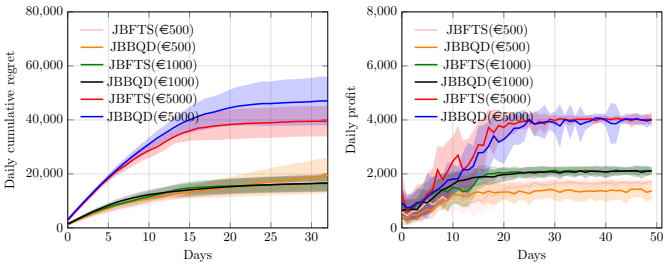

Figure 51: Arms=5. (left) Average daily cumulative regret with standard deviation intervals for Scenario 1.2.1. (right) Average daily profit with standard deviation intervals for Scenario 1.2.1. Last number in the legend indicates the cumulative daily budget invested.

References

[1] Agrawal, S., and Goyal, N. Analysis of thompson sampling for the multi-armed bandit problem. In *COLT* (2012).

[2] Auer, P., Cesa-Bianchi, N., and Fischer, P. Finite-time analysis of the multiarmed bandit problem. *Machine Learning 47* (2002), 235–256.

[3] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. The nonstochastic multiarmed bandit problem. *SIAM J. Comput. 32*, 1 (Jan. 2003), 48–77.

[4] Cao, Y., Wen, Z., Kveton, B., and Xie, Y. Nearly optimal adaptive procedure for piecewise-stationary bandit: a change-point detection approach. *ArXiv abs/1802.03692* (2018).

[5] Chen, W., Wang, Y., and Yuan, Y. Combinatorial multi-armed bandit: General framework and applications. In *International Conference on Machine Learning* (2013), pp. 151–159.

[6] Nuara, A., Trovò, F., Gatti, N., and Restelli, M. A combinatorial-bandit algorithm for the online joint bid/budget optimization of pay-per-click advertising campaigns.

[7] Rasmussen, C. E., and Williams, C. K. I. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[8] Trovò, F., Paladino, S., Restelli, M., and Gatti, N. Improving multi-armed bandit algorithms in online pricing settings. *International Journal of Approximate Reasoning 98* (2018), 196 – 235.

[9] Wikipedia contributors. Knapsack problem — Wikipedia, the free encyclopedia, 2020. [Online; accessed 17-April-2020].