Social Influence and Advertising

Online Learning Applications

Davide Grisi
Pasquale Scarmozzino
Michele Cosi

20th July 2022

Table of Contents

E-Commerce

E-Commerce Setup

- Let's consider a fictitious E-Commerce selling computers and peripherals, in particular the products offered are: (1)Computers at 1000$, (2)Monitors at 300$, (3)Headphones at 100$, (4)Keyboards at 75$ and (5)Mice at 30$. The Computer and Monitor will be called Tier1 products, while the rest Tier2 products.
- The site design is shown in Figure 1, so when a product is bought there are two suggestions or secondary products with different priority, the price of which is shown only when clicked.

Site Page

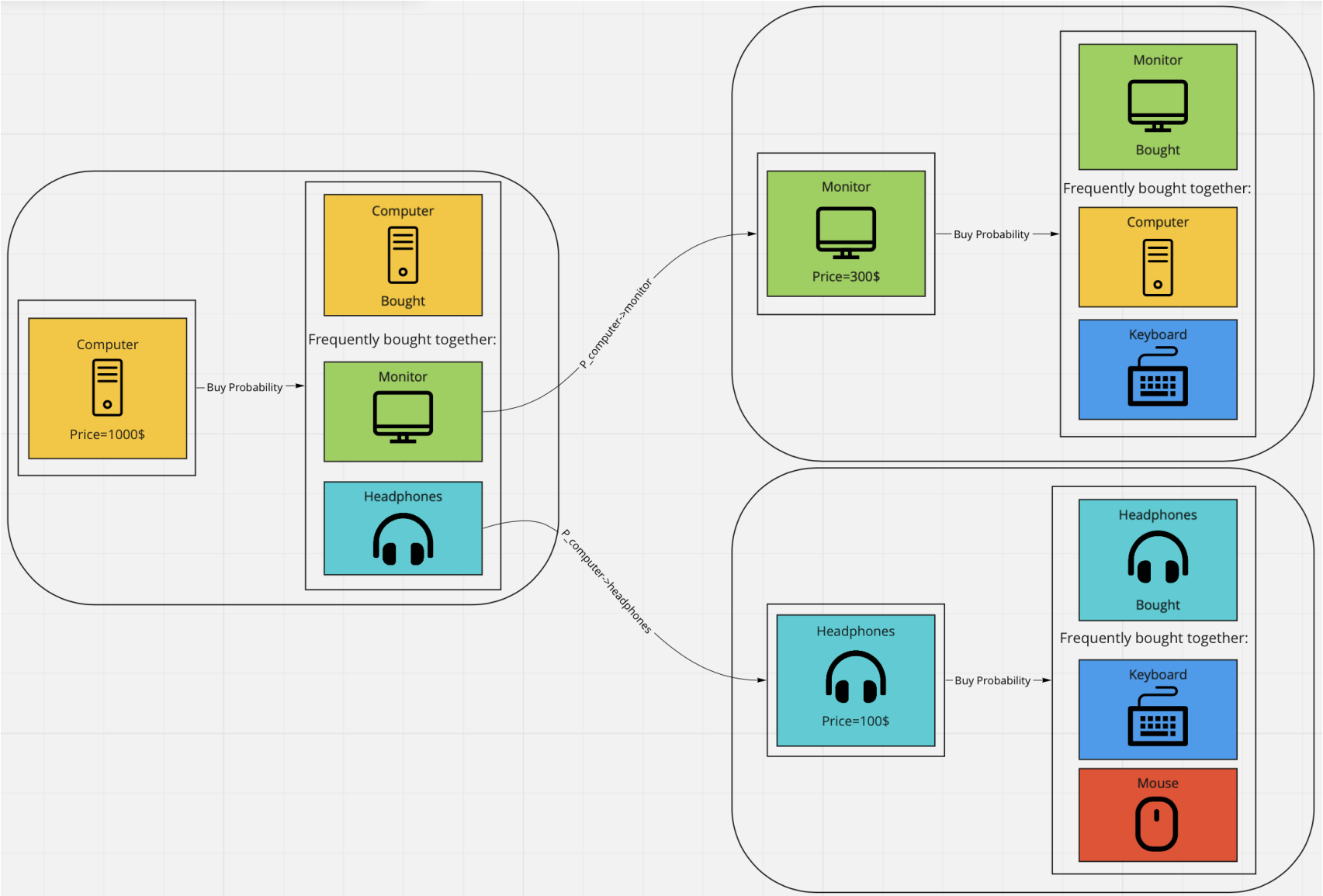


Figure 1: Site Structure

E-Commerce Setup

- The connection to first suggestion and second suggestion generate the graph shown in Figure 2. Considering that every node is associated with the initial page of the corresponding product, the probability that a user will for example end up on the Monitor page when previously on the Computer page will be given by the probability to buy the Computer times the probability to click on the Monitor suggested.
- The probabilities to click on a suggested product (without considering the probability to buy the primary product) can be represented with a matrix $[P_{ij}]$ which denotes the probability of clicking on the j-th product when suggested on the i-th page, the fact that the probability to click on a second suggestion is lowered by a factor λ will be implicit in the values of the matrix P.

Graph

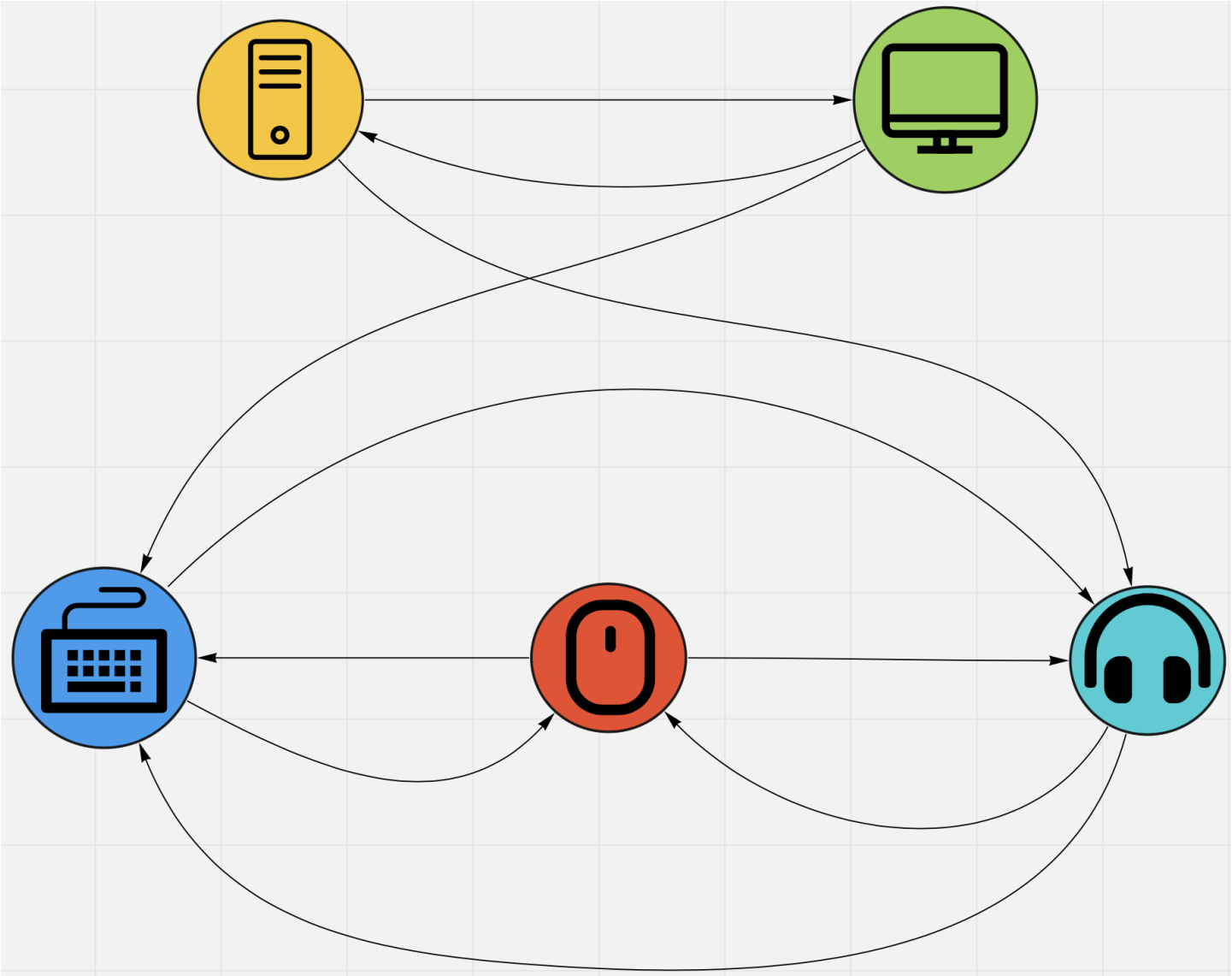


Figure 2: Graph generated from connection of primary to secondary products

Instance Chosen

Here we present the numerical values chosen:

$$P = \begin{bmatrix} 0 & 0.9 & 0.3 & 0.0 & 0.0 \\ 0.5 & 0 & 0 & 0.8 & 0 \\ 0.0 & 0.0 & 0.0 & 0.6 & 0.6 \\ 0.0 & 0.0 & 0.7 & 0.0 & 0.9 \\ 0.0 & 0.0 & 0.7 & 0.9 & 0 \end{bmatrix} \qquad \text{BuyProbability} = \begin{bmatrix} 0.8 \\ 0.5 \\ 0.9 \\ 0.7 \\ 0.3 \end{bmatrix}$$

$$\text{AverageSold} = \begin{bmatrix} 2 \\ 4 \\ 1.5 \\ 2 \\ 3 \end{bmatrix} \qquad \text{Margins} = \begin{bmatrix} 1000 \\ 300 \\ 100 \\ 75 \\ 30 \end{bmatrix}$$

Advertising Setup

Consider that every day a pool of new users enter the market. These users will visit our products' pages and competitors' pages in a percentage given by the α values. In particular, as shown in Figure 3, α_0 identifies the percentage lost to competitors and α_i the percentage landing on our i-th product. Typically α_0 is larger than any α_i meaning we are not in a condition of monopoly and, from the learning point of view, less data on the visits is available.

Users Distribution

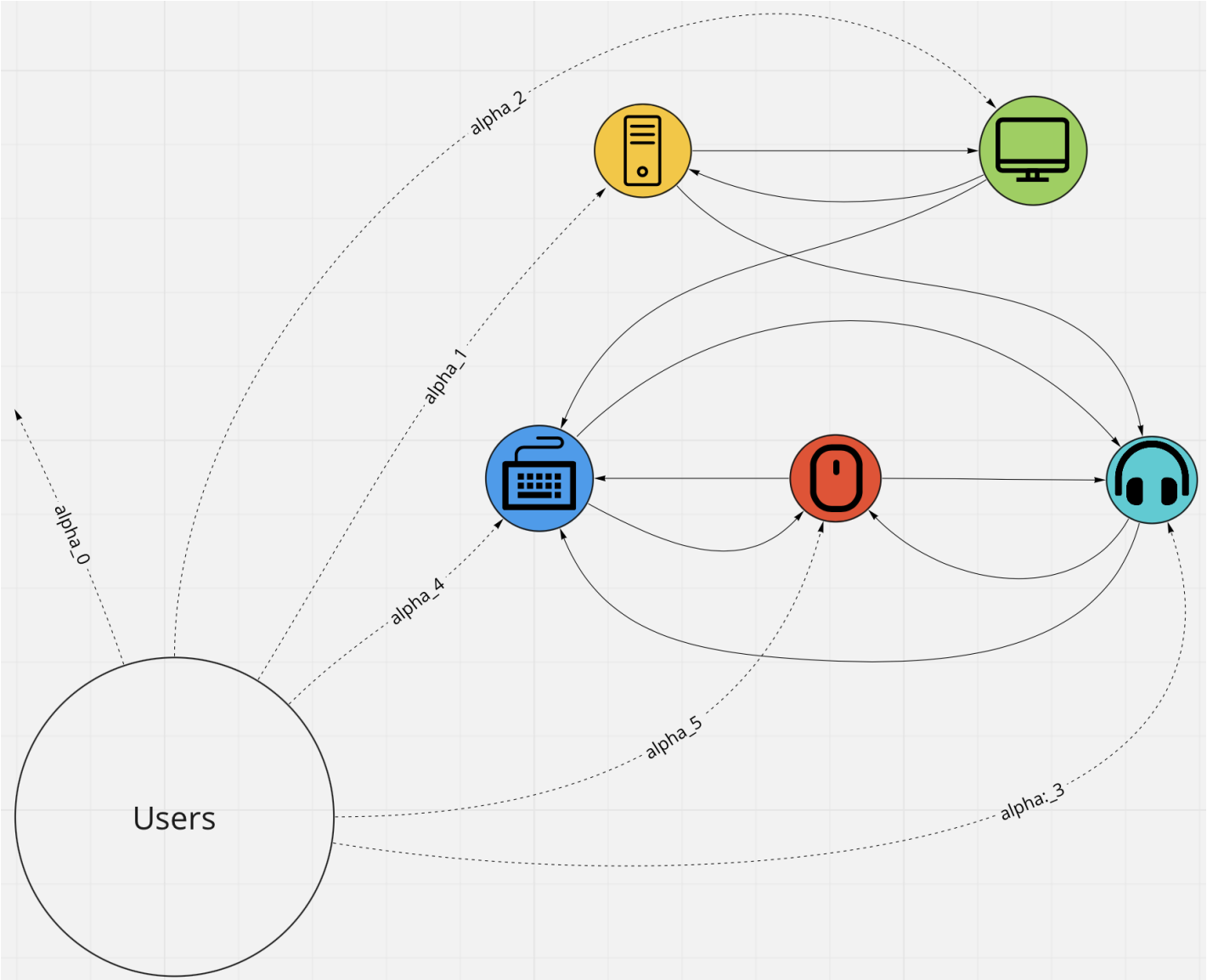


Figure 3: Scattering of the Pool of Users

Advertising Setup

- The α values are given by a Dirichlet distribution whose parameters are functions of the amount of money b_i that is invested in each product's advertising campaign.

$$(\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) \sim \text{Dirichlet}(\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5),$$

  where the parameter governing the proportion for the competitor is $\beta_0 = \text{total\_sum} - \sum_{i=1}^{5} \beta_i$. Total_sum chosen to govern the variance of the Dirichlet components.

- The beta-functions and thus the alpha-functions considered will be locally linear, and characterized by an indifference stage, a growing stage, and a saturated stage, an example is represented in Figure 4.

$$\mathbb{E}[\alpha_i] = \begin{cases} 0 & \text{if } b_i \leq b_i^{\min} \\ \frac{\bar{\alpha}_i (b_i - b_i^{min})}{b_i^{\max} - b_i^{\min}} & \text{if } b_i^{\min} < b_i < b_i^{\max} \\ \bar{\alpha}_i & \text{if } b_i \geq b_i^{\max} \end{cases}$$

Dirichlet Parameter as Function of budget spent

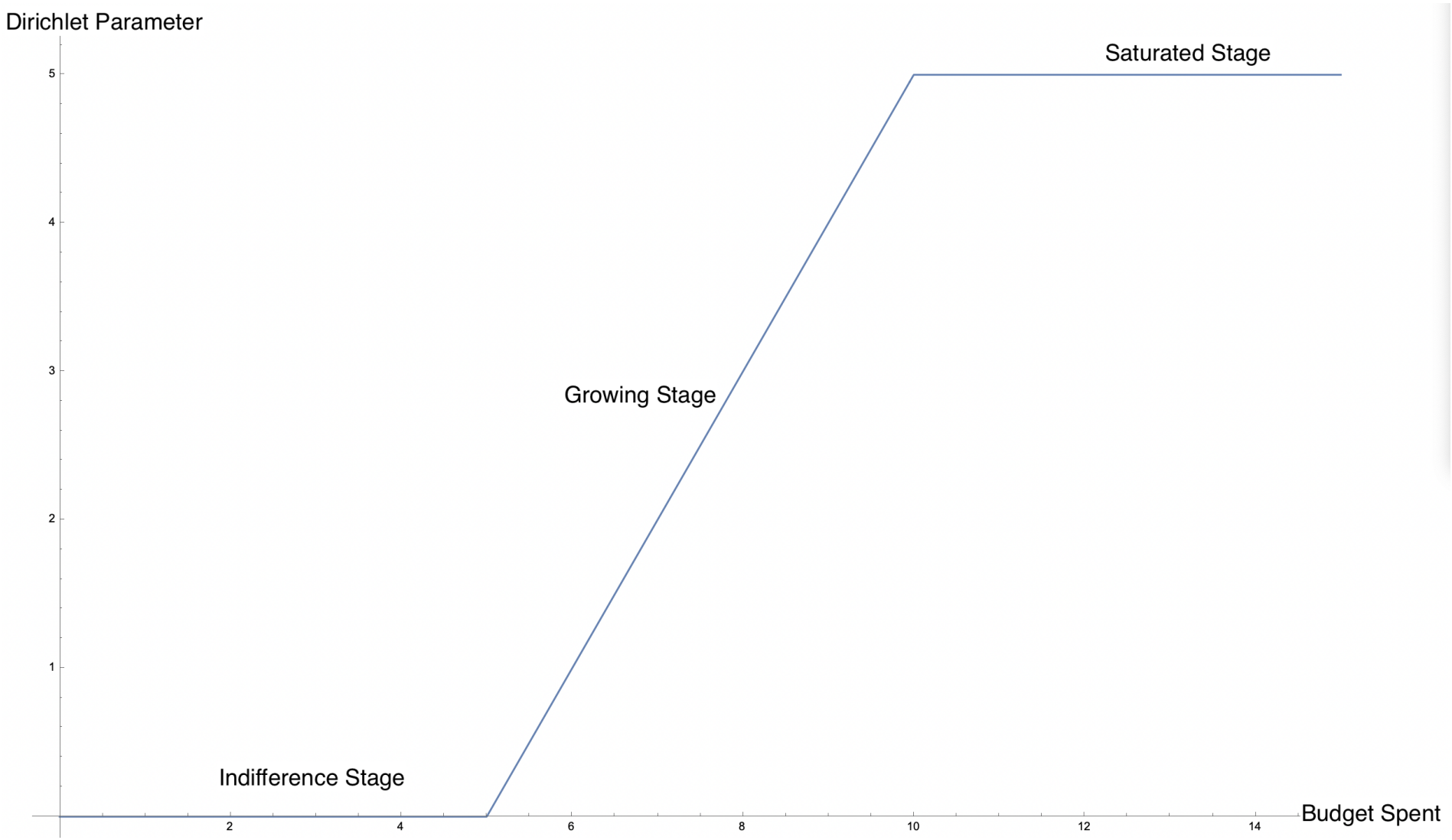


Figure 4: The functions considered will be locally linear and have three phases, indifference, growing and saturated

Instance chosen

The parameter list is (b_i^{min},b_i^{max},$\bar{\beta}_i$) and univocally identify a parameter function.
Given these parameters, $\bar{\alpha}_i = \frac{\bar{\beta}_i}{300}$

- In steps 3-4-5 have been chosen respectively for each product the parametric functions: [(0, 30, 150), (0, 25, 15),(5, 20, 30),(5, 40, 45),(5, 25, 60)]
- For step 6 the functions chosen gradually change in order to increase the landing probability on Tier1 products. This should simulate an expansion of the E-Commerce.
- For step 7 the features are:
  - The country of the user, in particular if it is considered poor or rich.
  - If the user is a retail or a company.

  and the company user are considered equal, both if they are in rich or poor countries.

Goal

Given an advertising budget B the task is to optimize the distribution of the budget between the different products' campaign in order to achieve the highest possible expected reward. Identifying as b^p the chosen arm of the p-th product, $\mathbb{E}\mathcal{R}$ the expected reward, and LB^p, UB^p the eventual bounds on the p-th product campaign, we have:

$$\begin{aligned} &\text{maximize } \left(\mathbb{E}\mathcal{R}(b^1, b^2, b^3, b^4, b^5)\right) \\ &b^1 + b^2 + b^3 + b^4 + b^5 \leq B \\ &LB^p \leq b^p \leq UB^p \qquad \forall p \end{aligned}$$

The reward for day j is given by $\mathcal{R}(b^1, b^2, b^3, b^4, b^5) = \sum_{i=1}^{5} \alpha_i n_j m_i$ where m_i is the expected margin from product i that is the average profit when i is the first landing product, n_j is the number of users visiting the website on day j, and α_i is the probability that a user receives an ad with the i-th product. To obtain m_i it must be considered the buy probability of the products given the landing on product i p_k^i, then for each product k this probability is multiplied times AverageSold[k] and Margins[k], the sum of the results gives m_i.

Step 1: Environment

The Environments

To simulate the behaviour of users visiting the site we have developed two Environment classes.

- The first one computes the expected values of all the quantities of interest, so it is used to develop optimization algorithms and Clairvoyant algorithms. Despite having combinatorial complexity, it is more efficient with few products and can be used to reduce the computation time.
- The second Environment is a Random one, which as the number of samples tend to infinity will approximate the first one. In the learning process it is used as simulator because it has variance, hence simulate better the real world.

"Exact" Environment

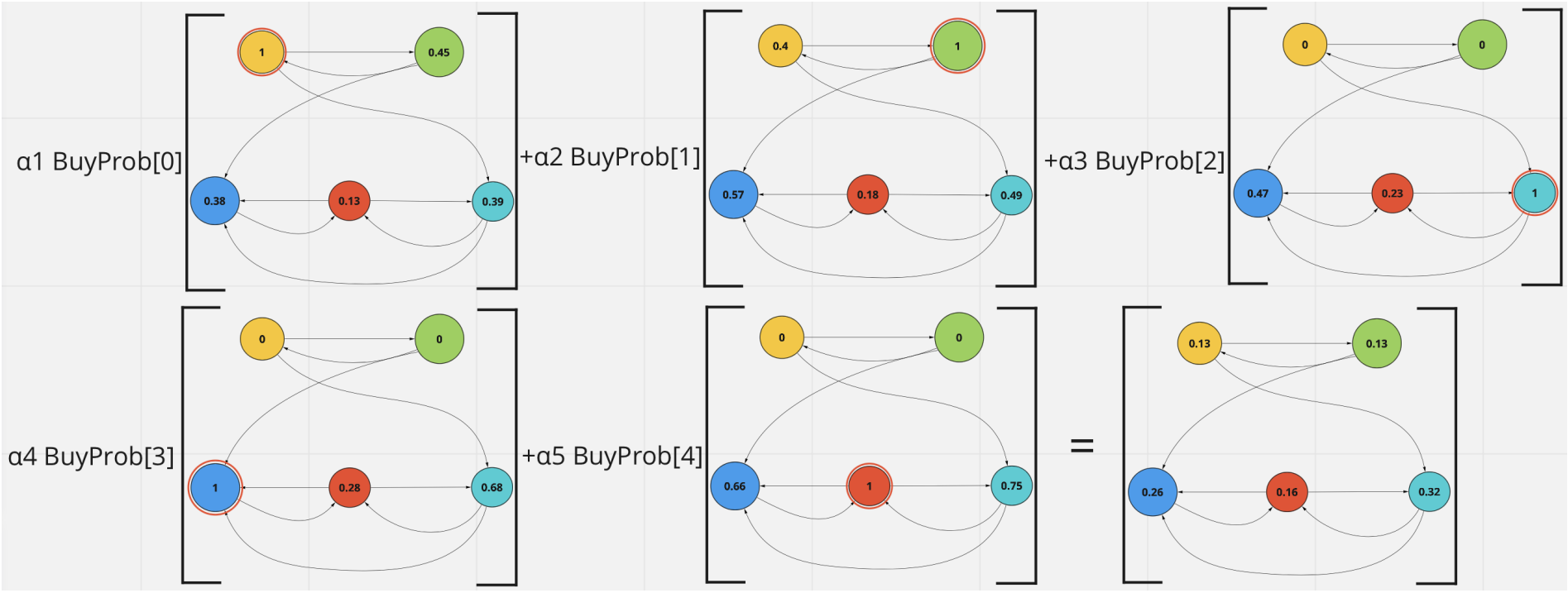
We call the "Exact" Environment the one that returns the expected values.

- The fundamental method of this class is the recursive function "site_landing" which given a landing node will calculate the probabilities of buying any other product. By landing it is meant that the user has bought the product and is on the suggestions page.
- Once the αs are known this method enable us to calculate the probability that a user buys each item, p_i. It is enough to multiply the "site_landing" distribution obtained for each product (which are the p_i^k of slide 13) by the associated α value and the respective probability to buy, and then sum over all the distributions obtained, that is $p_i = \sum_i \alpha_i p_k^i$. This process is represented in the figure of the example below.
- To calculate the expected reward per user of the selected αs the above equation is used. It must be multiplied p_i times the expected number of item sold times the margins of the product, than sum over all the products.

"Exact" Environment Algorithm Example

- First the distributions by landing (and buying) on each products are calculated using "site_landing": LandOn1=[1. 0.45 0.39 0.38 0.13] LandOn2=[0.4 1. 0.49 0.57 0.18] LandOn3=[0. 0. 1. 0.47 0.23] LandOn4=[0. 0. 0.68 1. 0.28] LandOn5=[0. 0. 0.75 0.66 1.]
- The probabilities are weighted with the α values and the probability to buy the corresponding product then are summed. In this example we will use $\boldsymbol{\alpha}$ =[0.5 0.1 0.1 0.15 0.05 0.1] obtaining a probability of buying over the nodes of $\boldsymbol{p}$=[0.13 0.13 0.32 0.26 0.16]. Multiplying these values by the number of users in the pool each day will hold the expected number of users that will buy each product.

"Exact" Reward



Having the probability of user purchase for each product it is trivial now to get the expected reward extracted per user. We just have to multiply each entry by the Average number sold and the Margin, than sum all the entries. In the example above this holds 373.25, which is the expected reward extracted from each user in the pool when the αs are given by $\boldsymbol{\alpha}$.

site_landing algorithm

Algorithm 1 site_landing: Recursive method for products purchase probability

Require: Connectivity matrix P, purchase probabilities prob_buy for each product
Require: landing product p, visited_nodes a (possibly empty) set of visited nodes

1: ret $\leftarrow [0, 0, 0, 0, 0]$
2: ret$[p] = 1$, in any case 1 is returned for the landing page
3: con_row$_j \leftarrow P_{pj}$, get exiting connections
4: masked_connection $\leftarrow$ con_row $\cdot$ visited_nodes, mask the connection with the activated nodes
5: visited_nodes$[p] \leftarrow 0$, the current node is deactivated
6: **if** masked_connection is all 0 no further connection can be explored **then**
7: **return** ret
8: **else if** masked_connection has only one value different than 0 in position q **then**
9: weight $\leftarrow$ con_row[q] prob_buy[q], probability to get to the connected product's suggestion page
10: **return** ret+weight*site_landing(q,visited_nodes), it's added the probability obtained from the connected node weighted accordingly
11: **end if**

site_landing algorithm continued

1: It is left the case in which there are two connected pages, with index q and k.
2: First considering only the cases in which ONLY a single one is visited
3: weight1 $\leftarrow$ (1 $-$ con_row[k]) con_row[q] prob_buy[q], only the first is visited
4: Visit1 $\leftarrow$ weight1 site_landing(q,visited_nodes)
5: weight2 $\leftarrow$ (1 $-$ con_row[q]) con_row[k] prob_buy[k], only the second is visited
6: Visit2 $\leftarrow$ weight2 site_landing(k,visited_nodes)
7: Second considering the cases in which both are visited
8: visited_nodes[k] $\leftarrow$ 0, the other node is deactivated because visited
9: weightBoth1 $\leftarrow$ con_row[k]con_row[q] prob_buy[q], both are visited and the first one is considered
10: VisitBoth1 $\leftarrow$ weightBoth1 site_landing(q,visited_nodes)
11: visited_nodes[k] $\leftarrow$ 1,visited_nodes[q] $\leftarrow$ 0 invert the visited
12: weightBoth2 $\leftarrow$ con_row[k]con_row[q] prob_buy[k], both are visited and the second one is considered
13: VisitBoth2 $\leftarrow$ weightBoth2 site_landing(k,visited_nodes)
14: **return** ret + Visit1 + Visit2 + VisitBoth1 + VisitBoth2

Landing probability

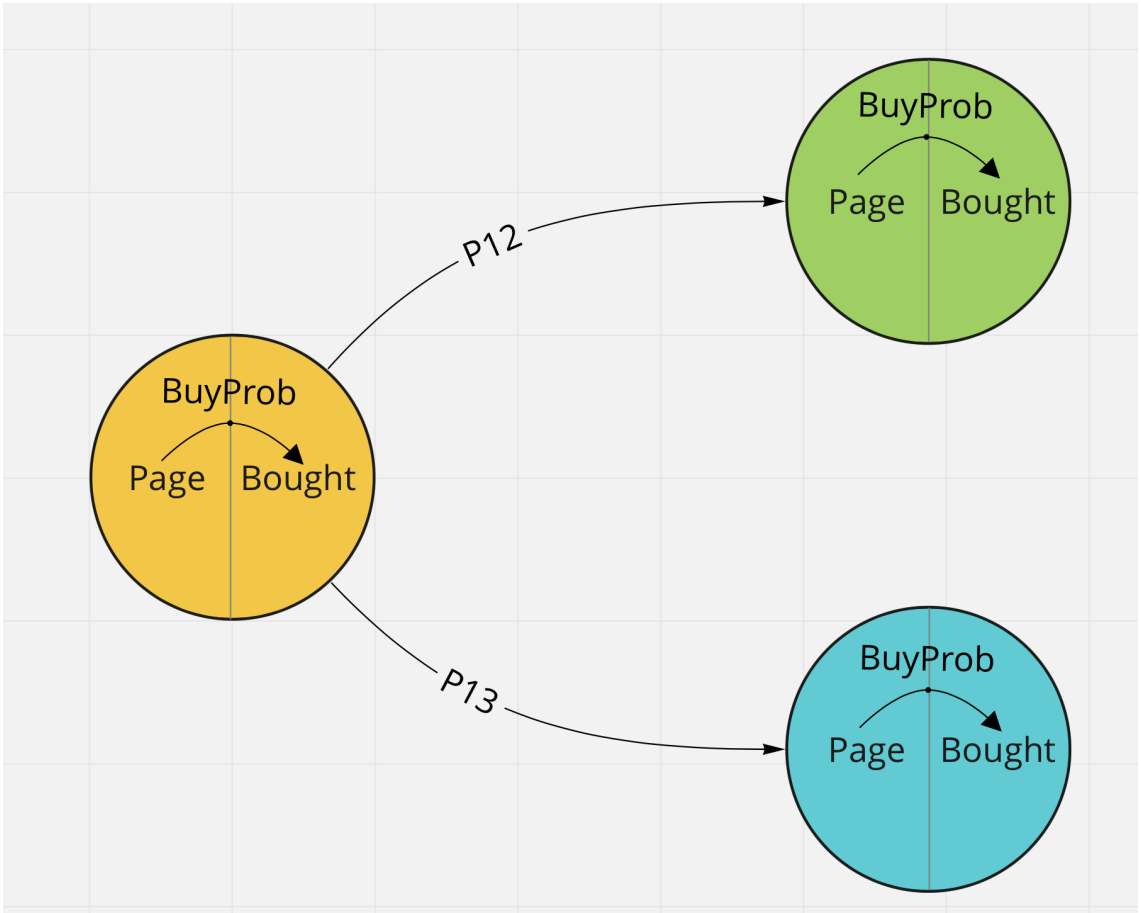


Figure 5: Given that the function will calculate the probabilities starting from the page of suggested products, when iterated recursively the successive result must be weighted by BuyProb times the probability of the scenario considered. So for example when considering the case in which only the first one is visited the probability of the scenario is given by $P_{12}(1 - P_{13})$

Random Environment

We call the Random Environment the one that simulates a real environment.

- it is composed by a series of user categories each distinguished by an average number of customers, features and different alpha-functions.
- at each round the superarm with the budgets allocated to each campaign (and the targeted features for step7) is provided. The budgets are split among the different user classes proportionally to their cardinality for the day sampled from a Poisson with class dependent mean and the alphas for the day.
- once the realizations of the Dirichlet are available, the visit of each user is simulated by assigning the first landing page according to a multinomial distribution with parameters alpha and the produced data (profit, number of items sold, visited pages, ...) is gathered and provided to the learners. The visit evolves recursively with a "site_landing" method different from the exact one which is explained in the next slides in pseudocode.

Random Environment, daily round

Algorithm 2 Algorithm representing a day in the RandomEnvironment

Require: an Environment with user classes $U_1, \ldots, U_k$, with mean number of users $\bar{n}_1, \ldots, \bar{n}_k$ and Dirichlet parameter functions $\mathcal{F}_1, \ldots, \mathcal{F}_k$

Require: a superarm (with targeting information if available) A

for $i = 1, \ldots, k$, for each user class U_i **do**
  $n_j^i \sim \text{Poisson}(\bar{n}_i)$, sample the number of users
  $\boldsymbol{\alpha}_i \sim \text{Dirichlet}(\mathcal{F}(A))$, find the alphas for current round
  for $u = 1, \ldots, n_j^i$, for each user **do**
    $p \sim \text{Multinomial}(\boldsymbol{\alpha}_i)$, find the landing product
    site_landing(p, visited_nodes $= \emptyset$, bought_nodes $= \emptyset$)
  end for
end for
return data generated during the visit (visited nodes, bought nodes, alpha realizations, profit for each class)

Random Environment, user visit or site_landing

Algorithm 3 Algorithm representing the visit of a user to the website

Require: An environment with connectivity matrix P and edge weights w_{ij}, purchase probabilities prob_buy for each product, demand for each product avg_sold

Require: landing product p, visited_nodes a (possibly empty) set of visited nodes, bought_nodes a (possibly empty) set of number of items bought

visited_nodes[p] $\leftarrow$ rec_level, use recursion level to establish the order of visit (for Step5)
if not Bernoulli(prob_buy[p]), (if not bought) **then**
    return
else
    bought_nodes[p] $\leftarrow$ Poisson(avg_sold[p])
    $p_1, p_2 \leftarrow \text{argmax}_{p_1,p_2} w_{p \to p_i}$, take secondary products
    if Bernoulli($w_{p \to p_1}$) and $p \notin$ visited_nodes **then**
        site_landing(p_1, visited_nodes, bought_nodes)
    end if
    if Bernoulli($w_{p \to p_2}$) and $p \notin$ visited_nodes **then**
        site_landing(p_2, visited_nodes, bought_nodes)
    end if
end if

Step 2: Optimization algorithm

Optimization algorithm

The main issue of advertising is that assigning a set of budgets $b_i \in \mathbb{R}^+$ to a set of subcampaigns c_i is a combinatorial problem whose exact solution is NP-hard (it is an instance of the knapsack problem). The formal statement of the goal is provided at slide 14.
In order to break the high complexity of the problem dynamic programming techniques are used:

- the space of the budgets is discretized,
- a $N_c x N_b$ (number of campaigns vs number of budgets) matrix is estimated containing the value each campaign provides when assigned a given budget,
- the maximum daily budget B is contained in the last column, bounds on the budgets for each campaign are reported in the initial matrix as $-\infty$,
- a solution is iteratively built starting from the first campaign to the last until all budgets are considered,
- eventually the budgets are subtracted from the last row entries and the maximum is taken,
- the algorithm both estimates the profit and provides the optimal allocation of budgets.

Optimization algorithm, certain α functions

The whole campaign has a maximum budget B and each subcampaign has upper and lower bounds on the budget assigned to it.
For the Optimization algorithm we defined a Value Matrix, whose elements w depend on the knowledge of the alpha-functions:
If the alpha-functions are certain the value w each subcampaign c provides when it is assigned a budget b and N users are available is given by

$$w_{cb} = \alpha_c(b) * N * v_c,$$

where v_c is the expected margin coming from landing on product c (or being targeted buy subcampaign c in Step7) and can be estimated either with the "exact" environment or the random one. This estimation implies in the random case the simulation of a large number of customer visits to get accurate values (> 1000 simulations). The complexity of the algorithm is given by $O(N_c N_b^2)$.

Optimization algorithm, certain α functions

Formally the objective is described by:

$$\text{maximize} \left(\sum_{i}^{N_c} \alpha_c(b_c) * N * v_c \right)$$

$$\sum_{i}^{N_c} b_c \leq B, \qquad \text{maximum budget constraint}$$

$$b_c^{\min} \leq b_c \leq b_c^{\max} \qquad \forall c = 1, \ldots, N_c, \qquad \text{single campaign constraints}$$

where $\alpha_c(b)$ is the "click through rate" for subcampaign c when budget b_c is allocated to it, N is the number of users and v_c is the expected value given by a user landing on the website following the subcampaign ad.

Optimization algorithm, pseudocode

Algorithm 4 Dynamic programming algorithm for budget allocation

Require: A $N_c x N_b$ matrix V with the values each campaign provides for every budget and $-\infty$ if the combination does not satisfy some upper or lower bound.
Require: $b_1, \ldots, b_{N_b}$ evenly splaced budgets from 0 to B=maximum daily budget.

$S \leftarrow \text{zeros}(N_c, N_b)$
solutions $\leftarrow \emptyset$
$S[0, :] \leftarrow V[0, :]$
for every campaign c **do**
  for every budget $b_i \in b_1, \ldots, b_{N_b}$ **do**
    $S[c, b_i] \leftarrow \max_{b_j \leq b_i} S[c-1, b_j] + V[c, b_i - b_j]$
    solutions$[b_i]$ = solutions$[b_j] b_i - b_j$, update solution for current budget.
  end for
end for
$S[c_{N_c}, :] - = [b_1, \ldots, b_{N_b}]$
opt_cell $\leftarrow$ argmax$(S[c_{N_c}, :])$
return $S[\text{opt\_cell}]$, solutions[opt_cell]

Step 3: Optimization with uncertain α functions

Optimization with uncertain α functions

We tackled the problem using two different ideas.

- The first idea is a data-greedy 5-dimensional online approximation (Using a 5D Gaussian Process) of the reward function. A super-arm is a choice of arm for each product, having five products a super-arm can be represented as a point in a 5-D grid generated by the arms of each product.
- The second idea is to approximate the α functions and pull the optimal superarm using the dynamic-programming algorithm given the approximated alphas as input.

Notice that the 5D GP will learn only from the reward of each superarm. Given n the number of arms there are n^5 possible superarms which live in a 5-dim space, so it is a really hard task for the 5D GP to estimate the reward in this space. Adding also that this learner does not leverage the known data about alpha and does not run any optimization algorithm, it's no surprise that this approach is really slow in learning.

Optimization with uncertain α functions

On the other hand the GP's on the α functions will use the information available and run an optimization algorithm. The main steps are:

- Approximate α-functions which are the only unknown
- Use the approximated αs with the Exact Environment to calculate the expected rewards. In particular the value matrix for the optimization algorithm is constructed.
- The dynamic-programming optimization algorithm extracts the best super-arm to pull.
- Gather the data and update the α-functions.
- Repeat.

This learner won't use in any way the information regarding the profit received, so the 5D one has been put on top of this in order to catch better arms nearer the approximately optimal one, but this worsened the results and was discarded.

Gaussian Processes

To reduce the amount of data required and leverage the correlation of the reward distribution of each arm, Gaussian Processes where used. GPs are a powerful model which can approximate every function $f(t)$ from data if some regularity conditions are satisfied.
Each arm is associated with a Gaussian distribution $\mathcal{N}(\mu_a, \sigma_a)$, whose parameters depend on the provided data and a kernel K defining the correlation among the arms. In our case the kernel is

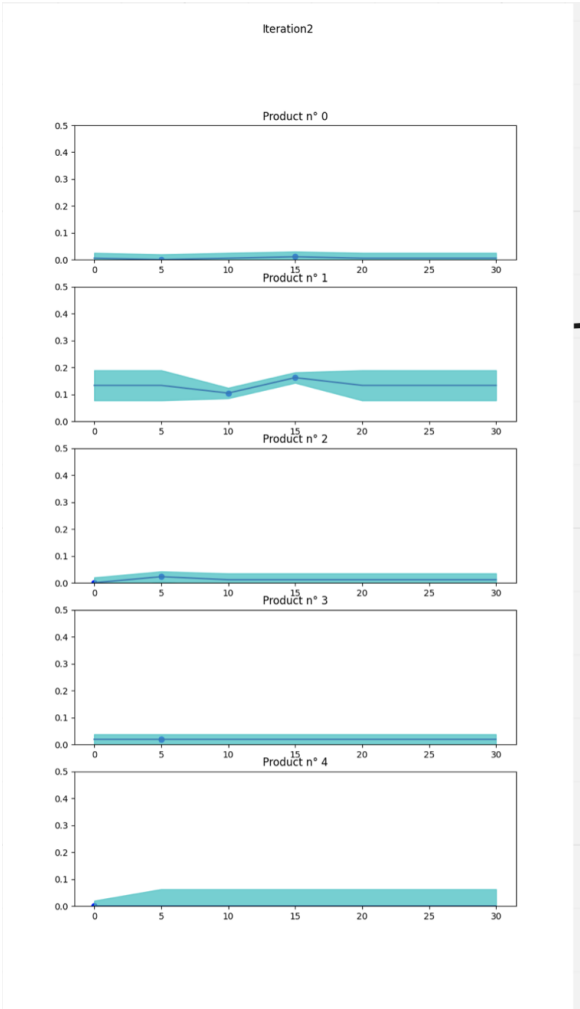
$$K(a, a') = C * \exp -\frac{||a - a'||_2^2}{2l^2} + W(s)$$

that is the sum between a Radial Basis Function kernel with C a multiplicative constant, l a "bandwidth" tempering the effect of the distance and W is a white noise with variance s.

Matching GP with TS and UCB

- In the case of GP combined with Thompson Sampling algorithm, we updated the elements of the Value Matrix using αs that are sampled from a gaussian distribution, that derives from the GP. Thus each *product* has a GP, and for each pair (*product,arm*) we drawn a sample from a normal distribution $\mathcal{N}(\mu_a^p, \sigma_a^p)$.
- Instead matching the GP with UCB, the α for each arm is estimated with an Upper Bound: after trying several UBs the best turned out to be $\mu_a^p + \sigma_a^p$, the sum between the mean and the standard deviation.
- for GPUCBs in the literature (Srinivas et al., Accabi et al. full reference at slide 95) there are different proposals weighting the standard deviation by $\sqrt{b_t}, t = 1, \ldots, T$, but for our case it turned out to provide worse performance in terms of regret. However, this choice prevents us from providing strong theoretical results on the regret bounds.

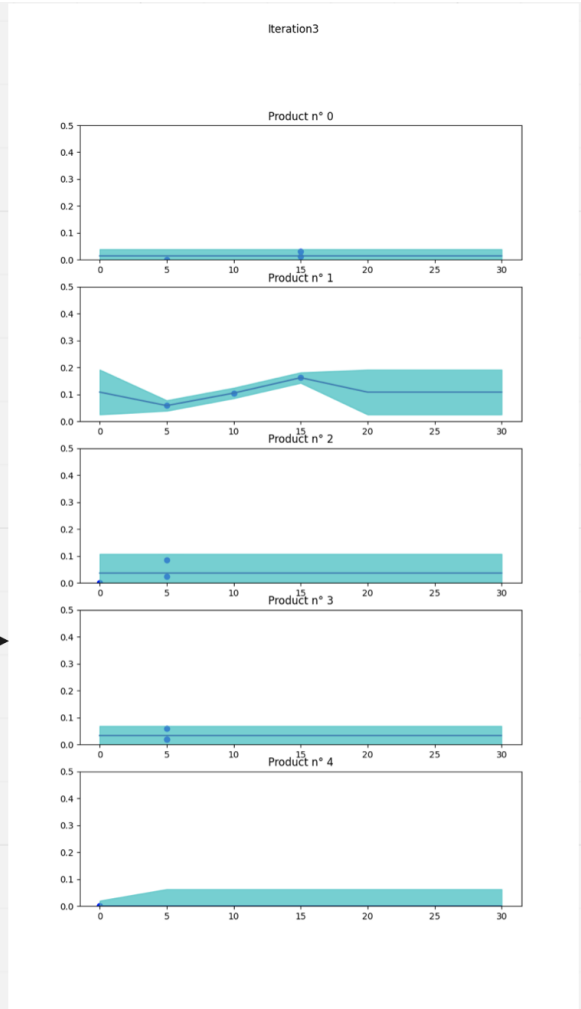
TS learner Process



Exact Environment:
Calculate Value Matrix

Dynamic-Programming Algorithm:
Best Super-Arm w.r.t. approximate alphas

Pull Super-Arm:
Get new data and fit the GP



Regret Result Step 3

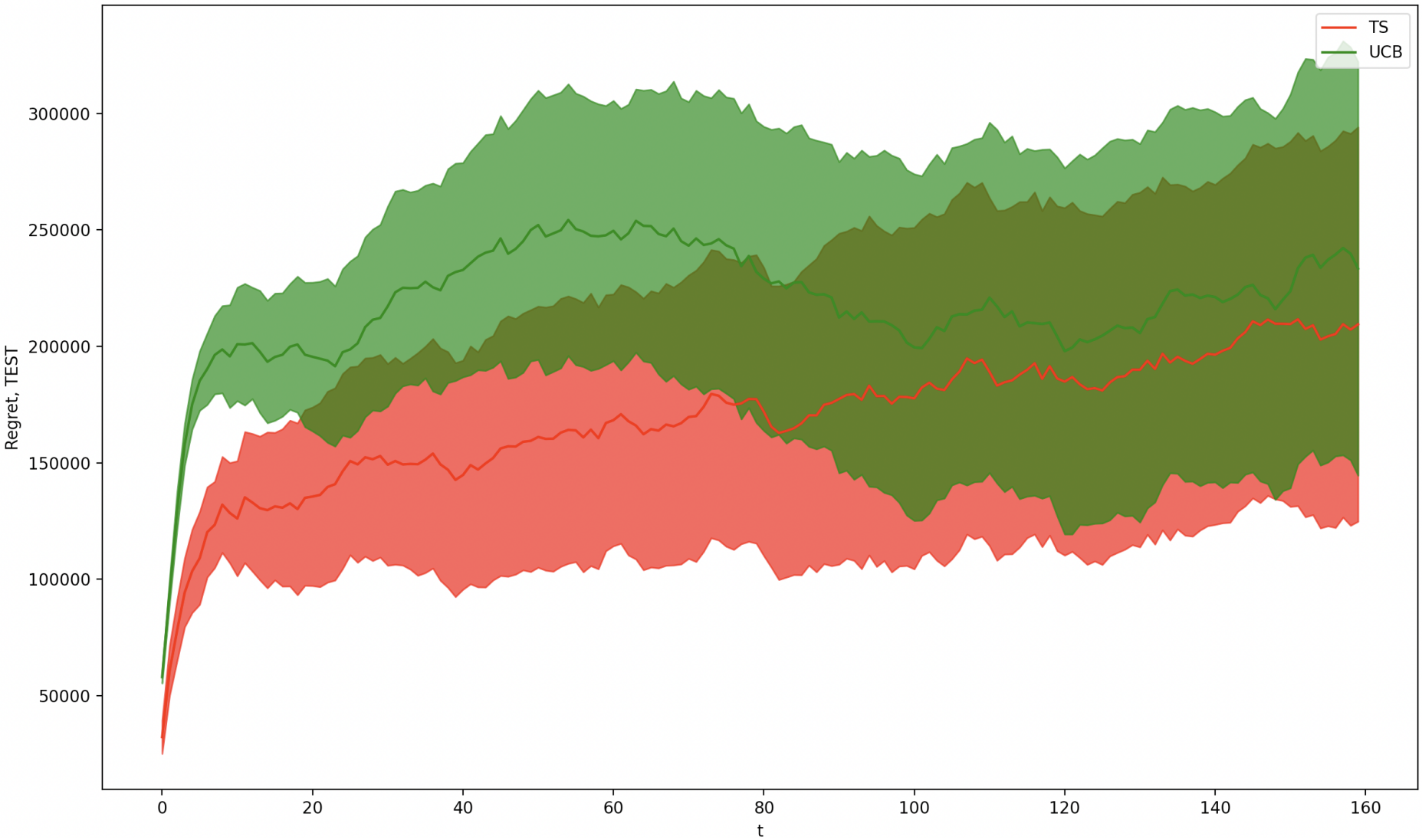


Figure 6: Regret of TS and UCB in Step 3. Results of 30 experiments

Reward Result Step 3

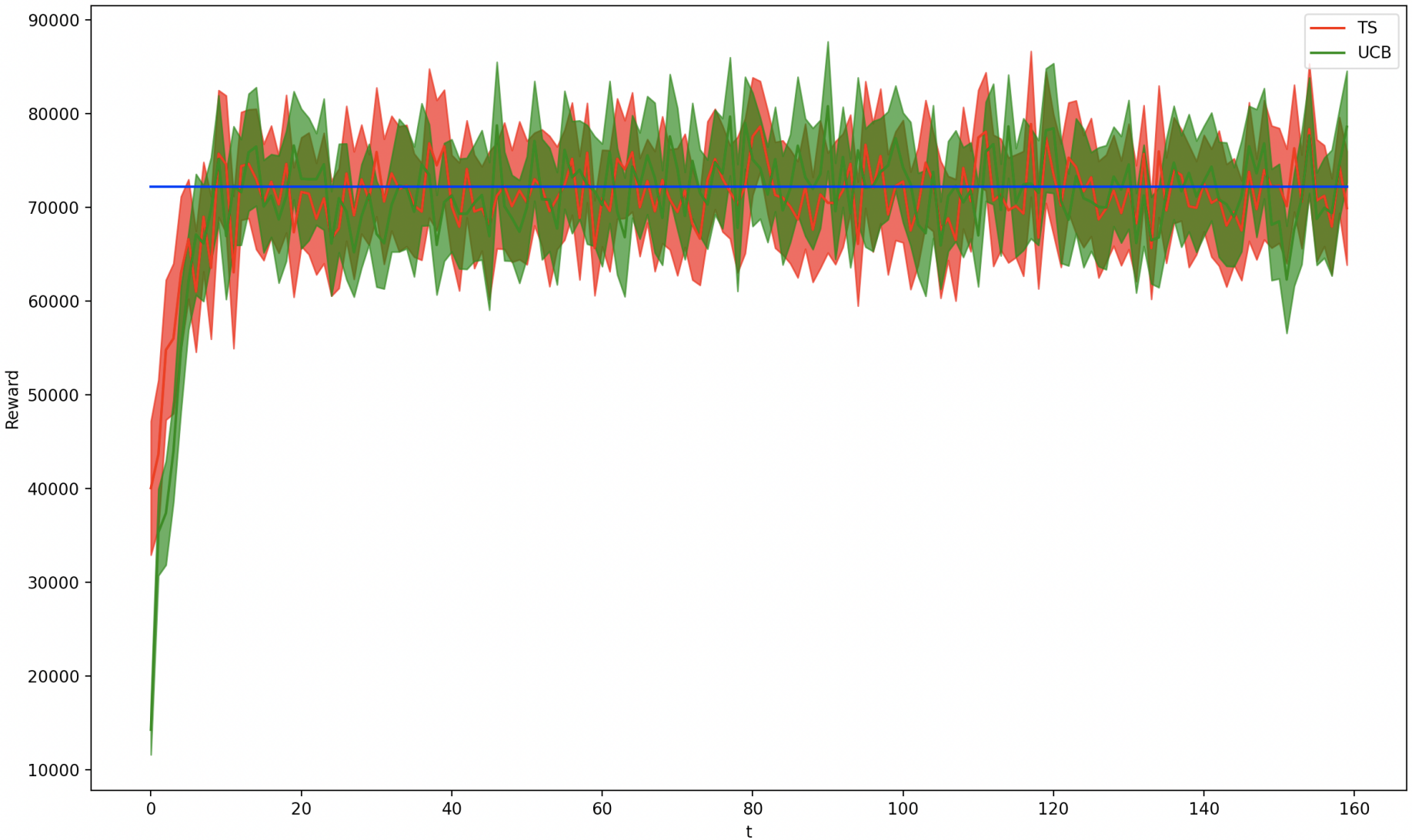


Figure 7: Reward of TS and UCB in Step 3 compared with optimal one, in blue

Regret Bound Step 3

The cumulative regret at time T is, with probability 1 - δ, smaller than the square root of some terms. Since the regret bound holds in probability it can happen that, with small probability, some experiments violate the bounds.

$$R_T \leq \sqrt{\frac{2\lambda^2}{\log\left(1+\frac{1}{\sigma^2}\right)} N_c T B \sum_{c=1}^{N_c} \gamma_{k,T}} \qquad \text{w.p.} \quad 1-\delta,$$

In detail, λ is the Lipschitz constant of the problem, σ is the maximum variance of the GP, the dependency on the number of GPs is denoted with the letter C, and $\gamma_{k,T}$ are the information gains at time T, that must be summed over all the GPs k. Finally, denoting the number of arms with M, B is defined as follows

$$B = 8 \log\left(2\frac{T^2 M N_c}{\delta}\right).$$

Following Srinivas et al. and Accabi et al. (full reference on slide 95), the sum of information gains $\sum_{c=1}^{N_c} \gamma_{c,T}$ is estimated with its upper bound given by $O(N_c(\log(TM))^{d+1})$, valid for Gaussian Processes with an RBF kernel, where d is the size of the arm.

Regret Bound Step 3, information gain and graph

The most important information in the graph is the trend, while the absolute value is just a rough estimate being the bounds in probability and being the combinatorial part an MC estimate. For UCB, our version has no theoretical results, weighting the variance in a different way w.r.t. the proposals in the literature.

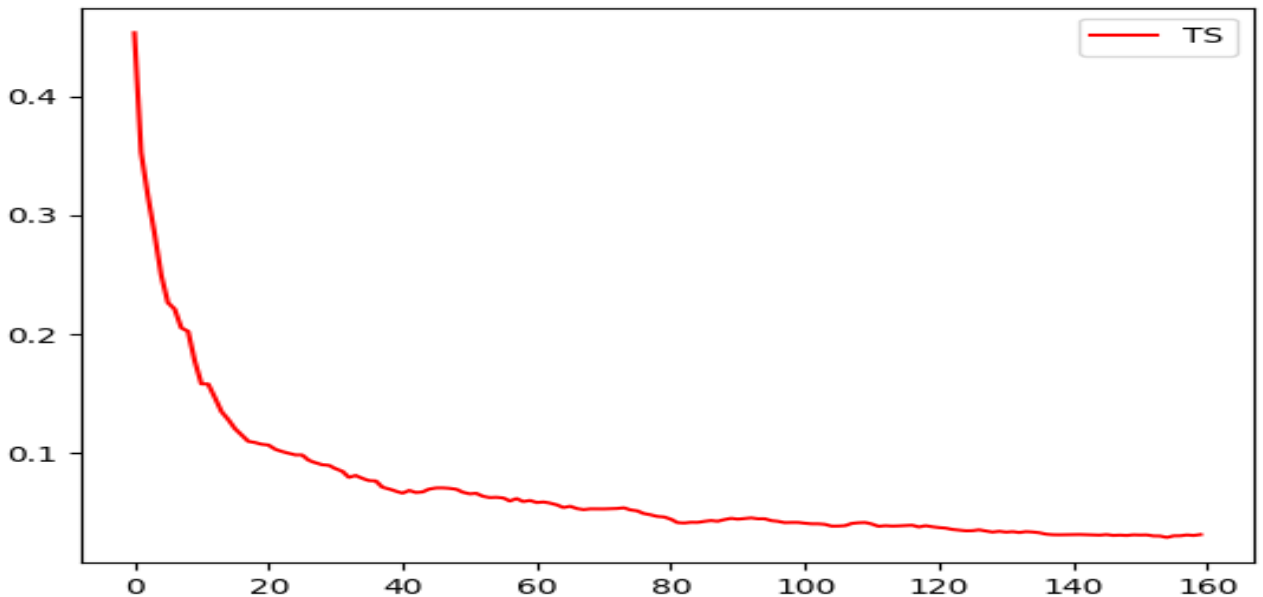


Figure 8: Ratio between mean regret and regret bound

Analysis and Comparison of the results

From the regrets plot (Figure 6) we notice two main differences between TS and UCB:

- It is clear that the TS is able to reach the optimal value in a more efficient way. In fact we notice that even if both seems to find the optimum in more or less 10 rounds (compare also Figure 7) at this round we see the TS with a mean regret between 100000 and 150000 while UCB stands around 200000.
- The second thing we notice is that going forward in time TS seems to keep increasing regret arriving at about 200000 at time 160 while UCB remains more or less stable at the same value. This is explained by the difference in variance of the two methods. Given that the UCB extracts the upper bounds it will more consistently pull the same arm, which once found will be the optimum. On the other hand the approximations of the TS are random normal variables which lower consistency, so even if the optimum has been found it may happen that a local maxima near the global one is pulled by TS due to its implicit variance.

Analysis and Comparison of the results continued

Another curious thing is the behaviour of the UCB regret which it seems it can decrease. To understand this behaviour we studied the distribution of the reward associated to the optimal superarm which is represented in Figure 12. From the histogram can be noticed how the distribution has an offset weighing on higher rewards, this explains why it is possible to have decreasing regret, as in the case of the UCB in this 30 experiments.
This consideration holds true also for the next steps and the non-fully connected case.

Reward distribution of optimal superarm

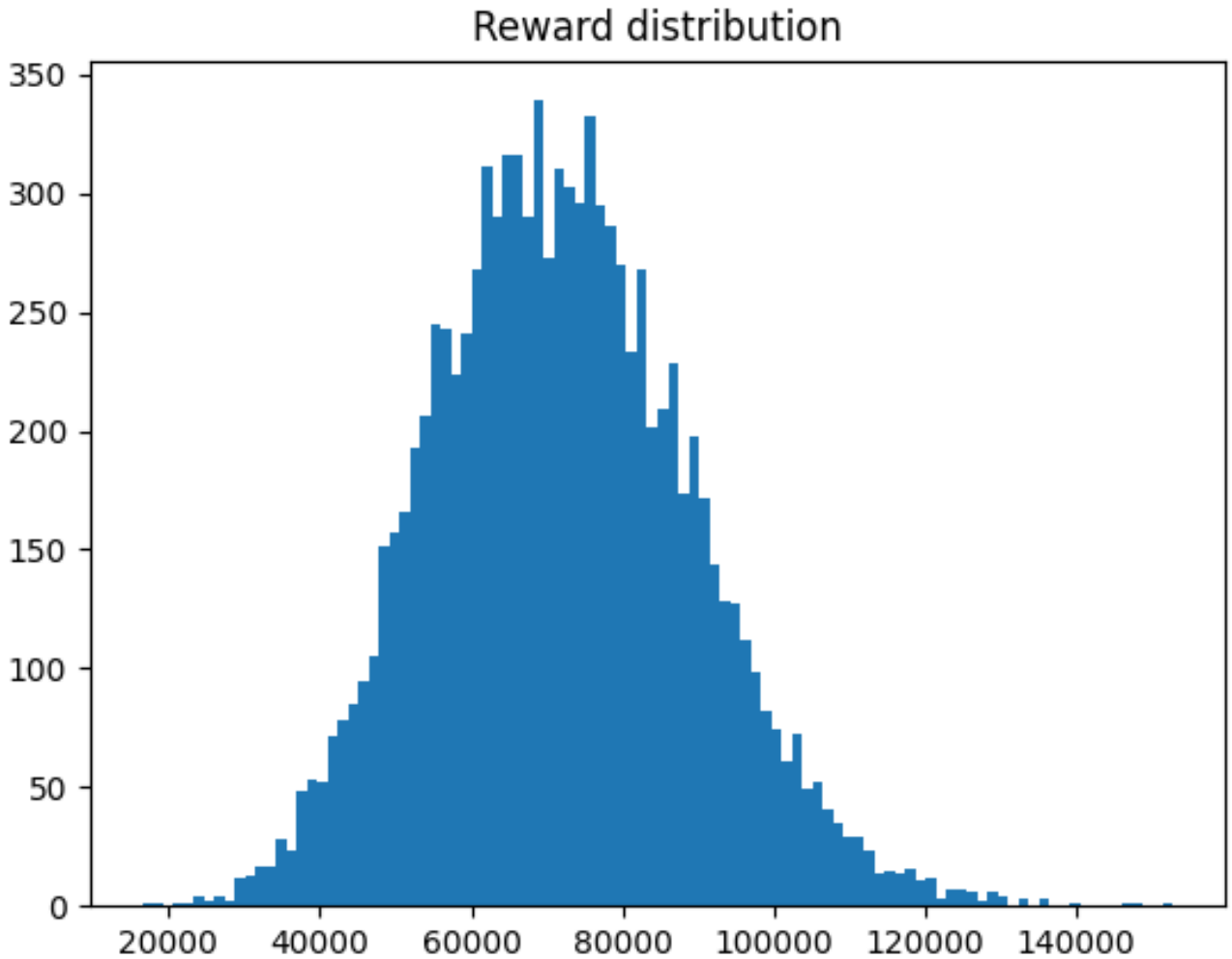


Figure 9: Distribution of the optimal super arm's reward, obtained by sampling the Random Environment. The distribution is slightly right-skewed and has an high variance.

Step 4: Optimization with uncertain α functions and number of items sold

Optimization with uncertain α functions and number of items sold

In this case also the number of item sold is unknown causing further uncertainty. Being the number of items independent samples from a product dependent $Poisson(\lambda_p)$, the MLE of the parameter λ of the distribution is given by the empirical average of the number of items sold N_p^v in every visit conditioned on the user buying the product during a website visit $N_{\text{visits buying p}}$.

$$\hat{\lambda}_p = \frac{\sum_{v \in Visits} N_p^v}{N_{\text{visits buying p}}}$$

The task is not much harder due to the rapid convergence of $\hat{\lambda}_p$, indeed as discussed in the analysis the order of magnitude of the reward is similar.

Regret Bound Step 4, information gain and graph

In this case additional regret terms should be added to the bound to include the error in estimating the parameters but they are hard to derive. So the usual bound was used just for **indication** to show the downward trend of the ratio.

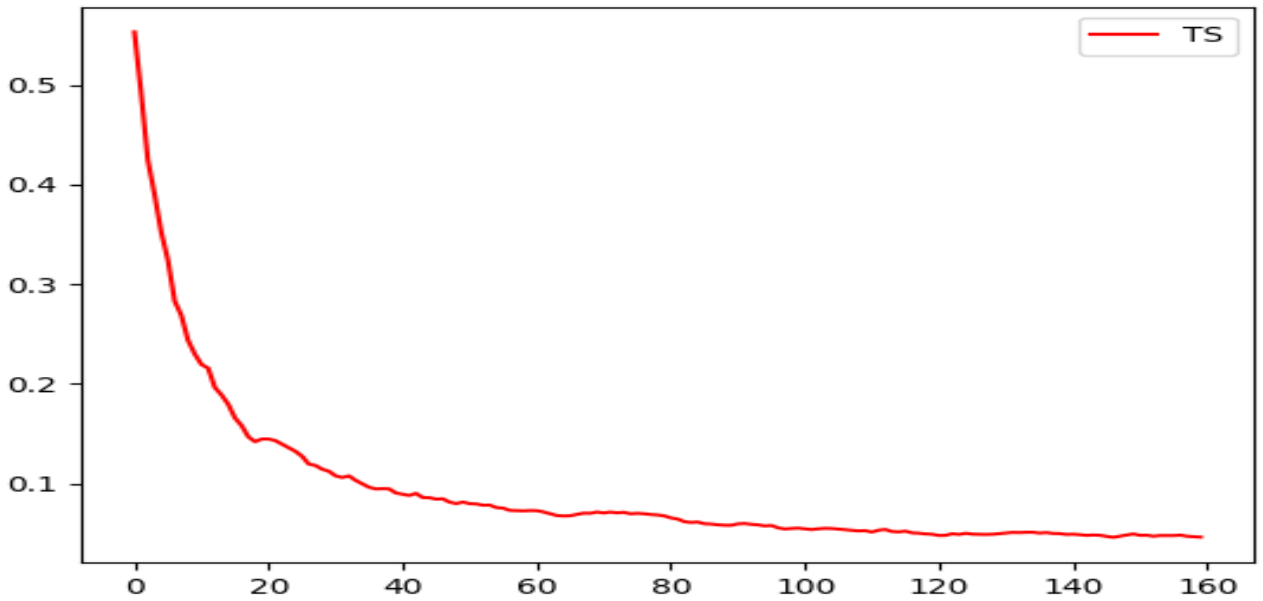


Figure 10: Ratio between mean regret and regret bound

Regret Result Step 4

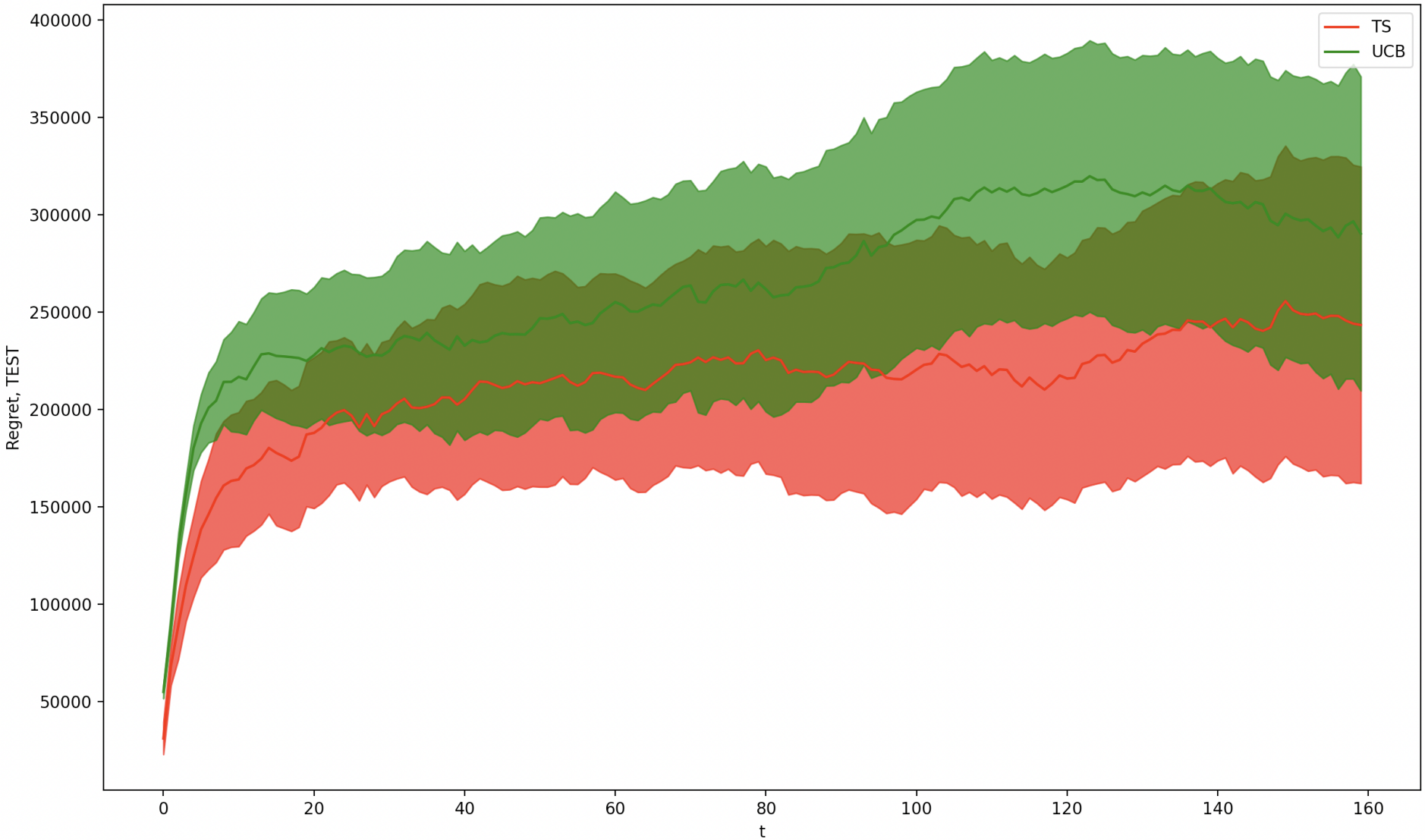
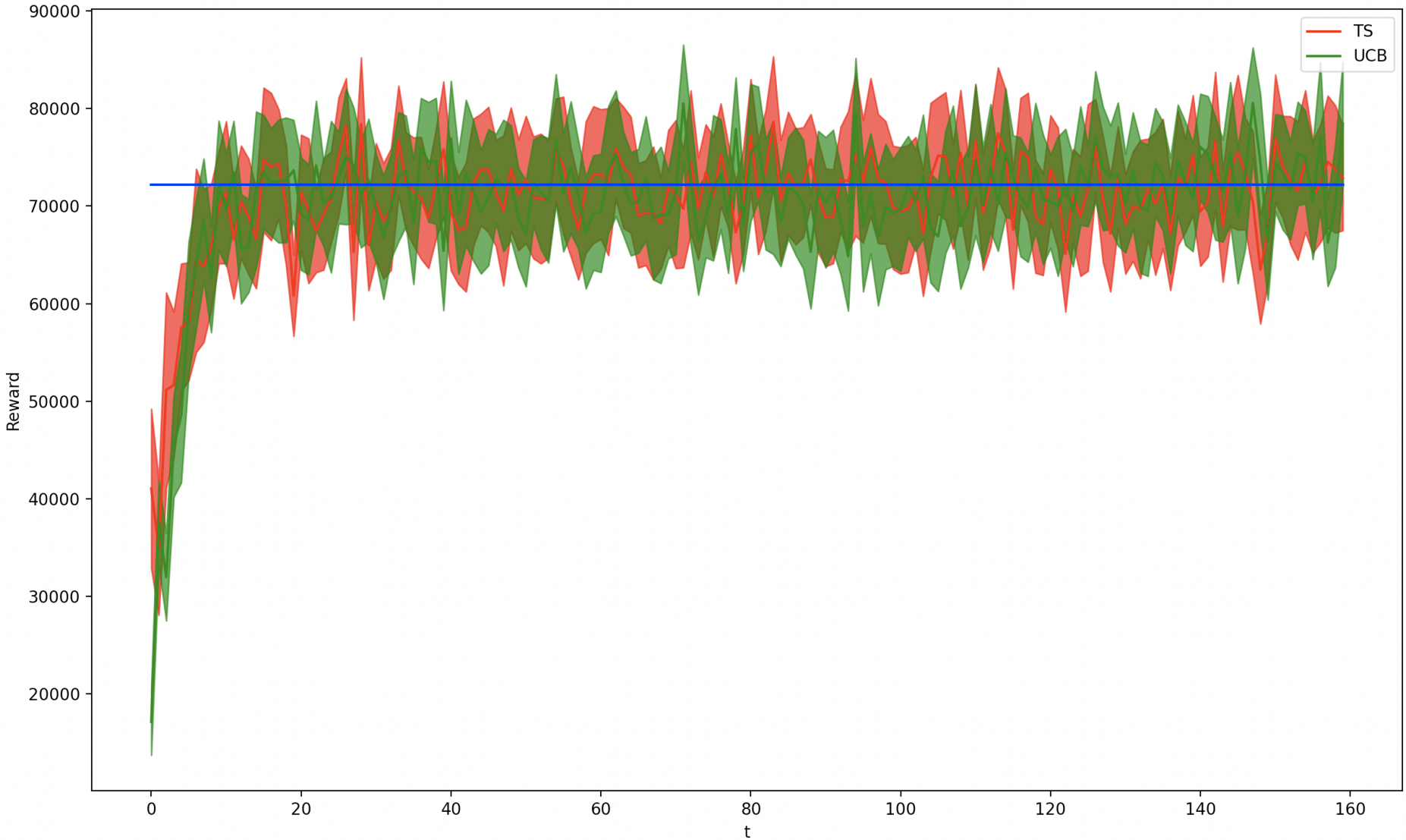


Figure 11: TS and UCB regret for step4, notice that the result is slighlty higher than that of step 3

Reward Result Step 4



Analysis of the results

As expected, the regret of the step 4 is slightly bigger than the one of the previous step. We also have higher variance due to the uncertainty of the numbers of items sold, in addition to the uncertainty of the α functions.
Moreover, the average number of items sold converges very quickly, thus it does not affect significantly the overall complexity of the optimization problem.

Step 5: Optimization with uncertain α functions and graph weights

Optimization with uncertain α functions and graph's weights

For this step also the graph's weights are unknown, increasing further the uncertainty.
The algorithm used to approximate the weights is the same as the one in youtube video exercise class with some correction to take into account whether an item has been actually bought or not. Indeed, a naive application of the algorithm would estimate for every edge the joint probability of buying the start product and then transitioning to the next P(user buys p $\cap$ user moves from p to p'), while, since we assume that the conversion rates are known, we just need to estimate the probability of transition once an item has been bought.
This case is slightly harder than the previous one and requires more use of social influence techniques. The website is assumed to know only the product pages visited by any user and not the exact transition from a page to the next.

Optimization with uncertain α functions and graph's weights

The Idea of the Algorithm used on each product is:

- For each node consider the connected nodes that have been visited in the previous time steps
- The estimate of the probability of each edge is computed by dividing the sum of the credits by the number of episodes in which the starting node either has been active previously than the node of interest or has been active and the ending node has not been active.
- Credits are given by assigning equal weight to the candidate previous nodes.
- Only the nodes-product that have been visited and purchased are considered as "previously active nodes" since the conversion rates are assumed to be known.
- the activation history is derived by the environment by storing in a vector the distance from the landing page of the current page.

$$p_{v,w} = \frac{\sum_{t=1}^{T} credit_{v,w}^{t}}{A_v}$$

$$credit_{p,p'}^{v} = \frac{1}{\sum_{w=1}^{N_p} I\left(t_w^v = t_p^v - 1\right)}$$

Edge weight estimation

Algorithm 5 Social influence algorithm to learn the transition probability P from one node to the next.

Require: A mask M s.t. $M[p, p'] = 1 \iff p \to p'$ is a legal transition.
Require: A previous count of the credits C and a normalization matrix A.
Require: A batch of visits V.

for every $v \in V$ **do**
  active_nodes $\leftarrow$ v.activated_nodes **and v.bought_nodes**
  for every product p **do**
    $t_p^v \leftarrow$ v.activated_nodes[p], get activation time for product p (0 if not active)
    previous_nodes $\leftarrow$ v.activated_nodes $= t_p^v - 1$**and** active_nodes
    C[previous_nodes, p]$+ = 1/|$previous_nodes$|$, assign the credit c
    $A[p'! = p, p] \leftarrow$ (active_nodes **and**$t_p^v = 0$)**or** previous_nodes, compute normalization factor adding one to the other nodes if active before p or active and p not active
    $P \leftarrow C[M]/A[M]$, compute the final probabilities for legal transitions
  end for
end for

Regret Bound Step 5, information gain and graph

In this case additional regret terms should be added to the bound to include the error in estimating the parameters but they are hard to derive. So the usual bound was used just for **indication** to show the downward trend of the ratio.

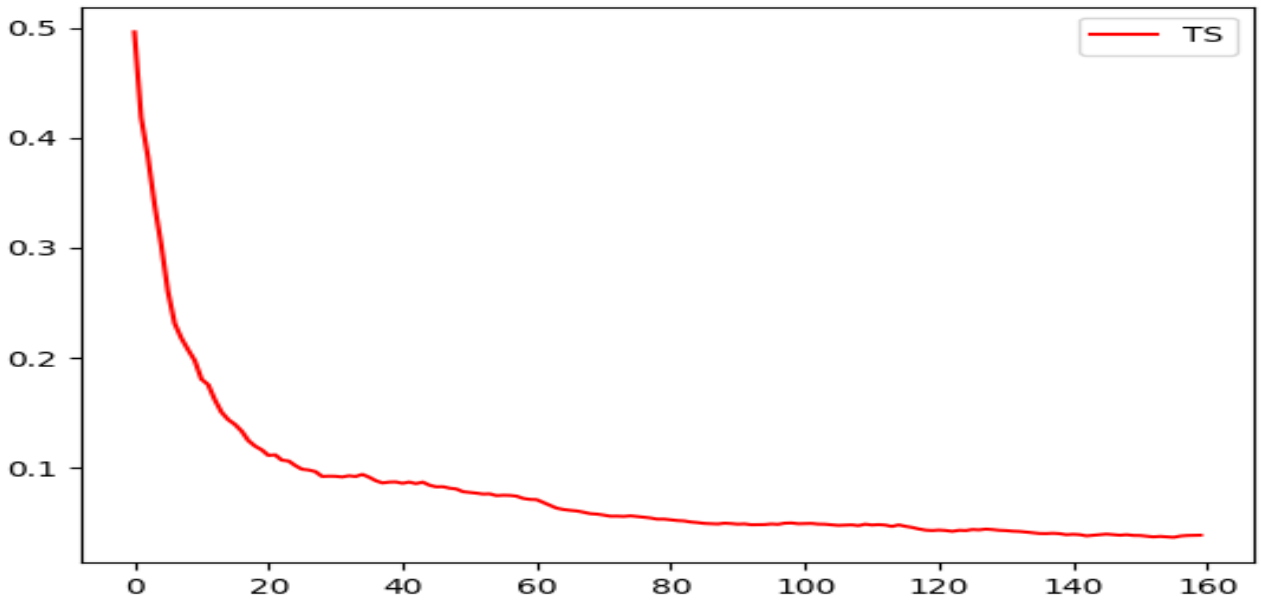


Figure 12: Ratio between mean regret and regret bound

Regret Result Step 5

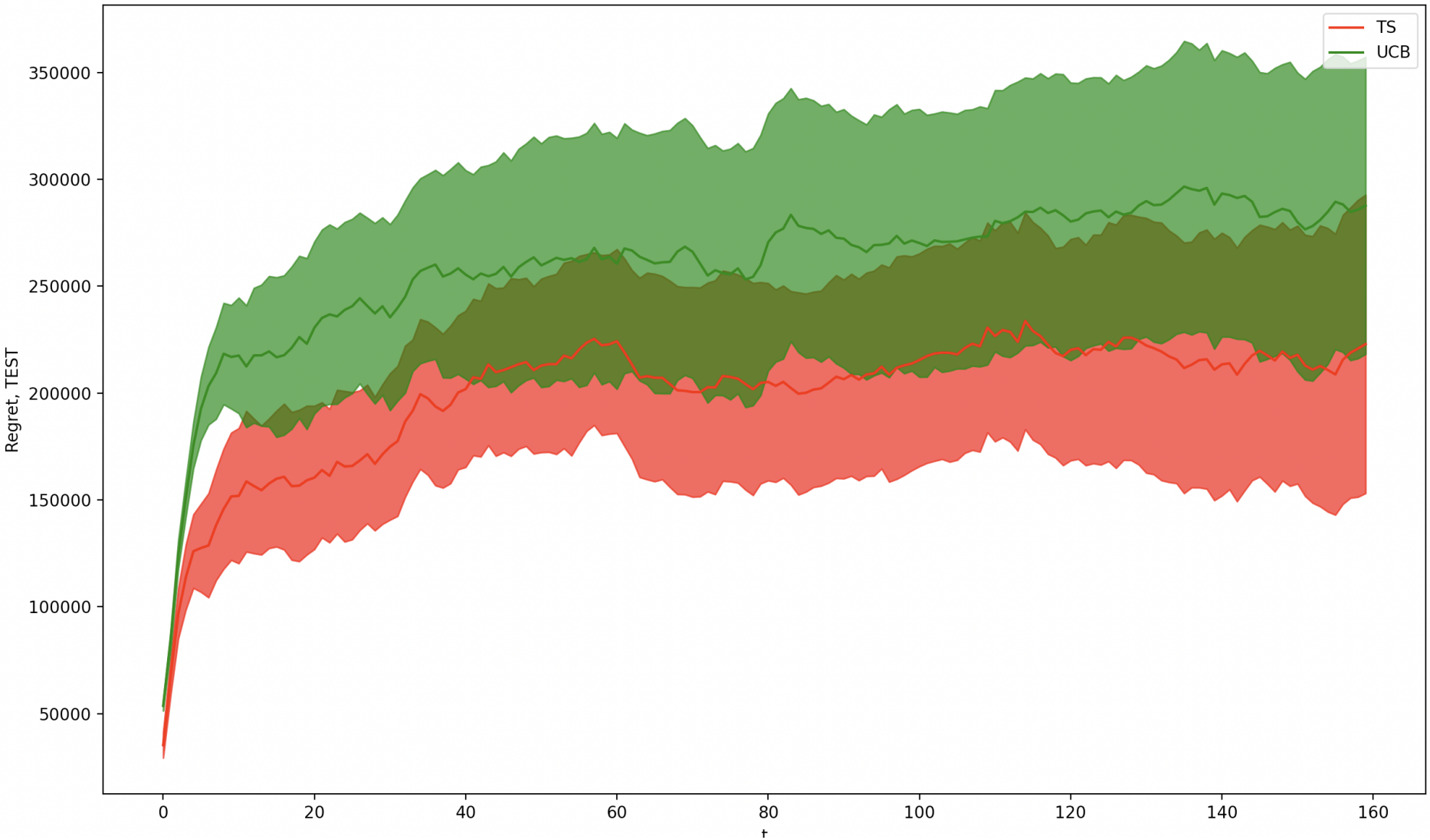
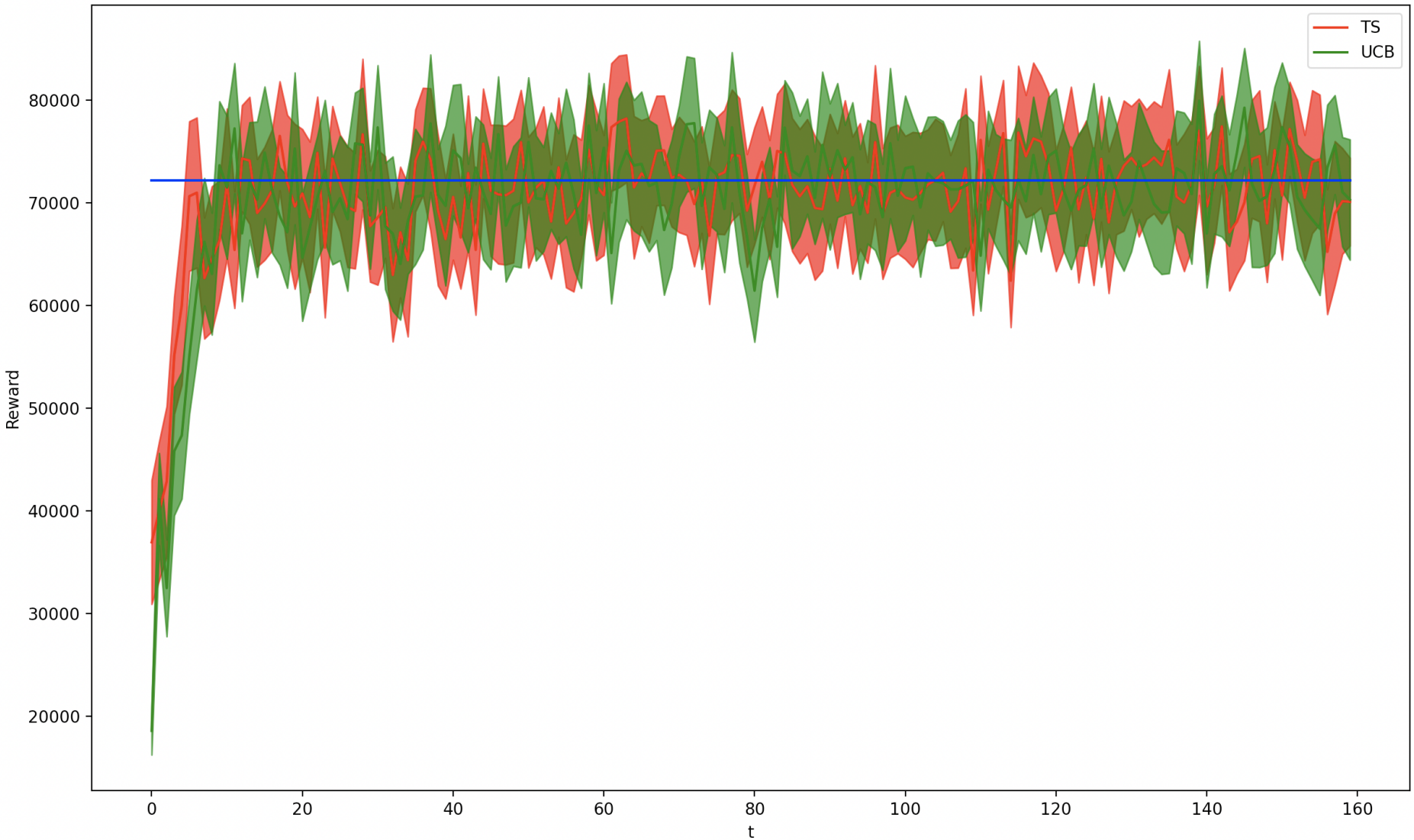


Figure 13: It starts to get flatter at day 40, that is the time it take the weight to converge

Reward Result Step 5



Analysis and Comparison of the results

Comparing the regret with the one of step 4 we notice that in this step it flatten only after the 40^{th} day, against at around time 10 for step 4. So we can infer that it takes 40 steps for the approximate weight to converge. Apart from this difference the regret does not seem to be any higher than in the previous case, nor having higher variance.

Step 6: Non-stationary demand curve

Non-stationary demand curve

To manage a non-stationary environment two methods have been implemented:

- A sliding window approach, in which not all the past data are considered, in particular the data older than $\tau = 2\sqrt{T}$ are discarded during the fitting of the GP.
- A learner provided with a change detection algorithm, in this case when a change is detected all the previous data are discarded, so the learner is discharged of wrong data from previous stages. When a change is detected there is a refractory period of 10 steps so that the GP stabilizes and false change detections are avoided.

Details on change detection

The Gaussian Process will generate a Normal distribution for each arm of the products. When a super-arm is pulled the data obtained can be compared with the Normal distribution estimated in order to identify eventual changes in the environment.
At every step we gather a sample for each product hence 5 independent Gaussian distributions are considered and the test can be done in two ways:

- Considering the 5 Normal distributions as a single 5-D Normal vector we can test for changes by looking at the p-value of the norm of the standardized 5-D vector of samples, this is analogous to imposing elliptical bounds on the vector. Using this method a change affecting every product's demand curve is more easily detected, while a change of demand on a single product need to be more relevant to be detected.
- Otherwise we could consider separately each product's sample and test every p-value. This approach is ideal to identify single product's changes while is less able to detect smaller global changes. In the 5-D setting this case in analogous to considering parallelepipedal bounds.

Different Bounds

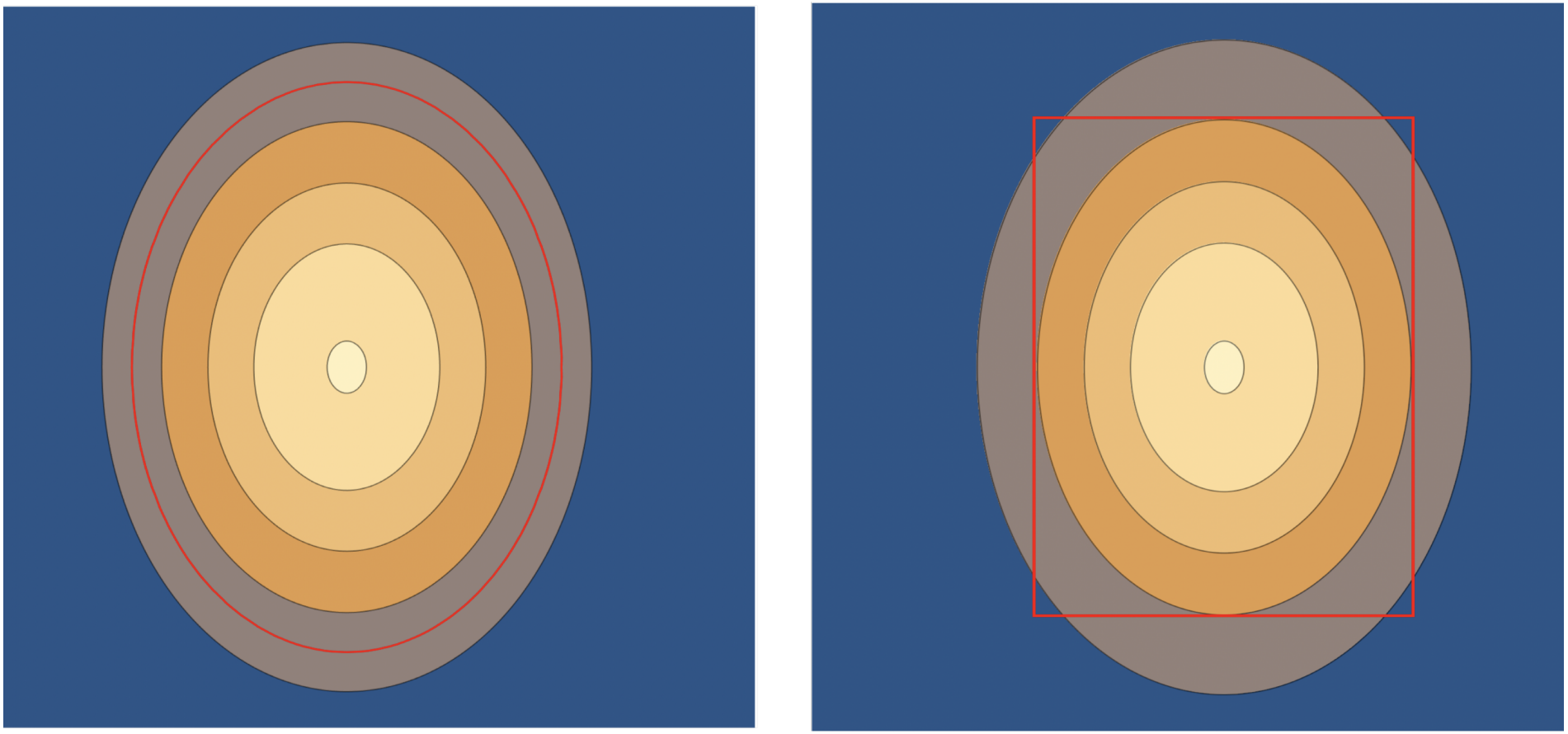


Figure 14: On the left elliptical bounds, used for considering an aggregate p-value. On the right parallelepipedal bounds, each edge is a 1D Gaussian bound

Details on change detection: 5D Bounds

Denoting with $\boldsymbol{x}$ the sampled vector we can standardize it obtaining $\boldsymbol{z}$ with $z_i = \frac{x_i - \mu_i}{\sigma_i}$ where μ_i and σ_i are the GP mean and standard deviation of the i-th product.
Having now $\boldsymbol{z}$ distributed as a standard 5D Gaussian we have to find the confidence bounds on the euclidean norm of the vector. In particular we detect a change if the norm of the sampled standardized vector has a p-value lower than β. First we find the probability of having norm less than P, the surface of the n-1 dimensional sphere is given by $\sigma_{n-1} = \frac{2\pi^{n/2}}{\Gamma(n/2)}$:

$$\text{Prob}\left[\, |Z| < P \,\right] = \frac{1}{(2\pi)^{n/2}} \sigma_{n-1} \int_0^P e^{-\frac{\rho^2}{2}} \rho^{n-1}\, d\rho = \frac{2^{1-\frac{n}{2}}}{\Gamma\left(n/2\right)} \int_0^P e^{-\frac{\rho^2}{2}} \rho^{n-1}\, d\rho$$

and the sought β-bounds are given by posing $\text{Prob}\left[\, |Z| < P \,\right] = 1 - \beta$ and solving numerically for P. So a change is detected if $|z| > P_\beta$.

Mathematica Code for Calculation of P_β

Asking for $1 - \beta = 0.99$ we get:

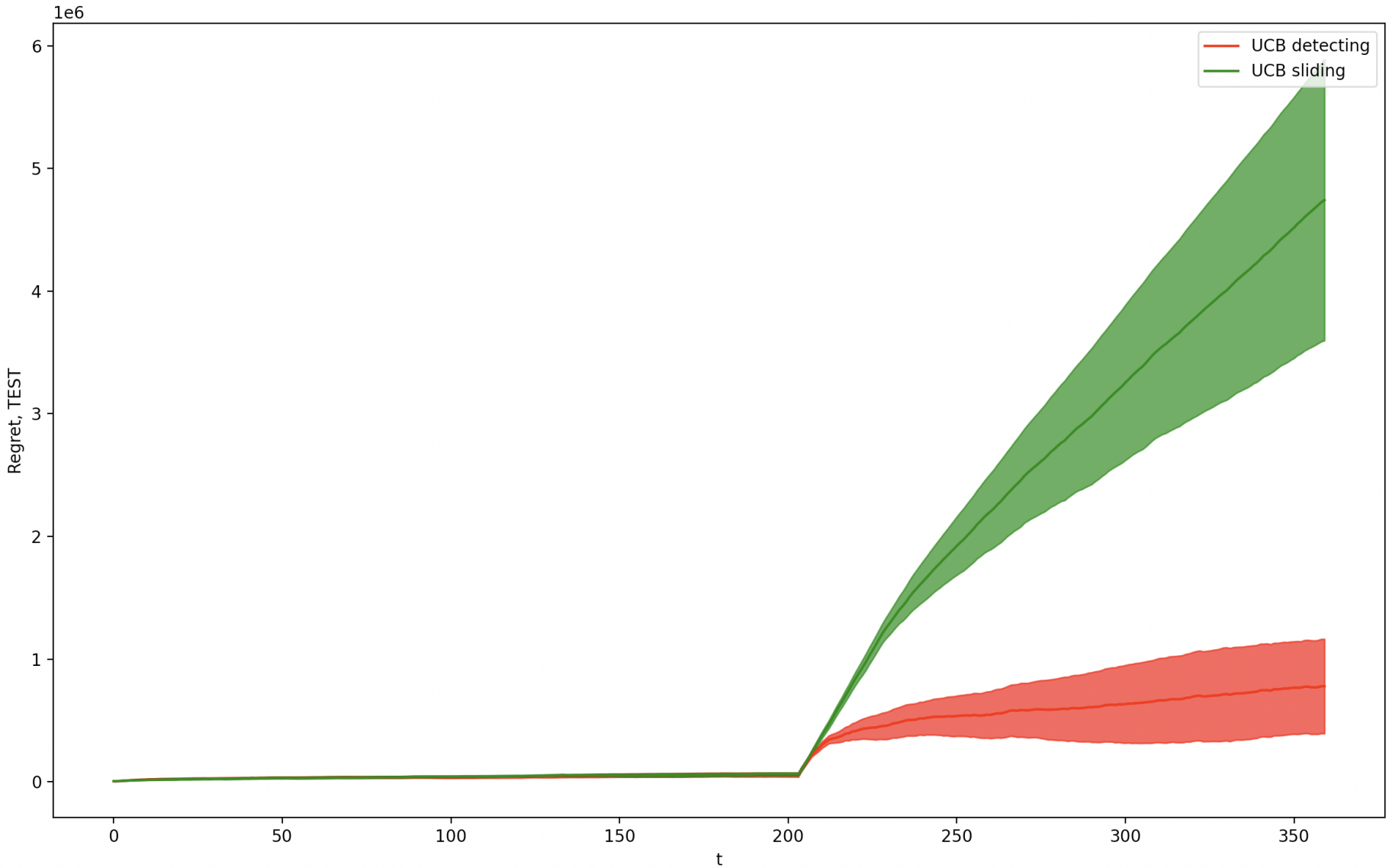
$$n = 5;$$

$$\text{Distribution[P\_]} = \frac{2}{\text{Gamma}[n/2]2^{n/2}} \int_0^P e^{-\rho^2/2} \rho^{n-1} d\rho;$$

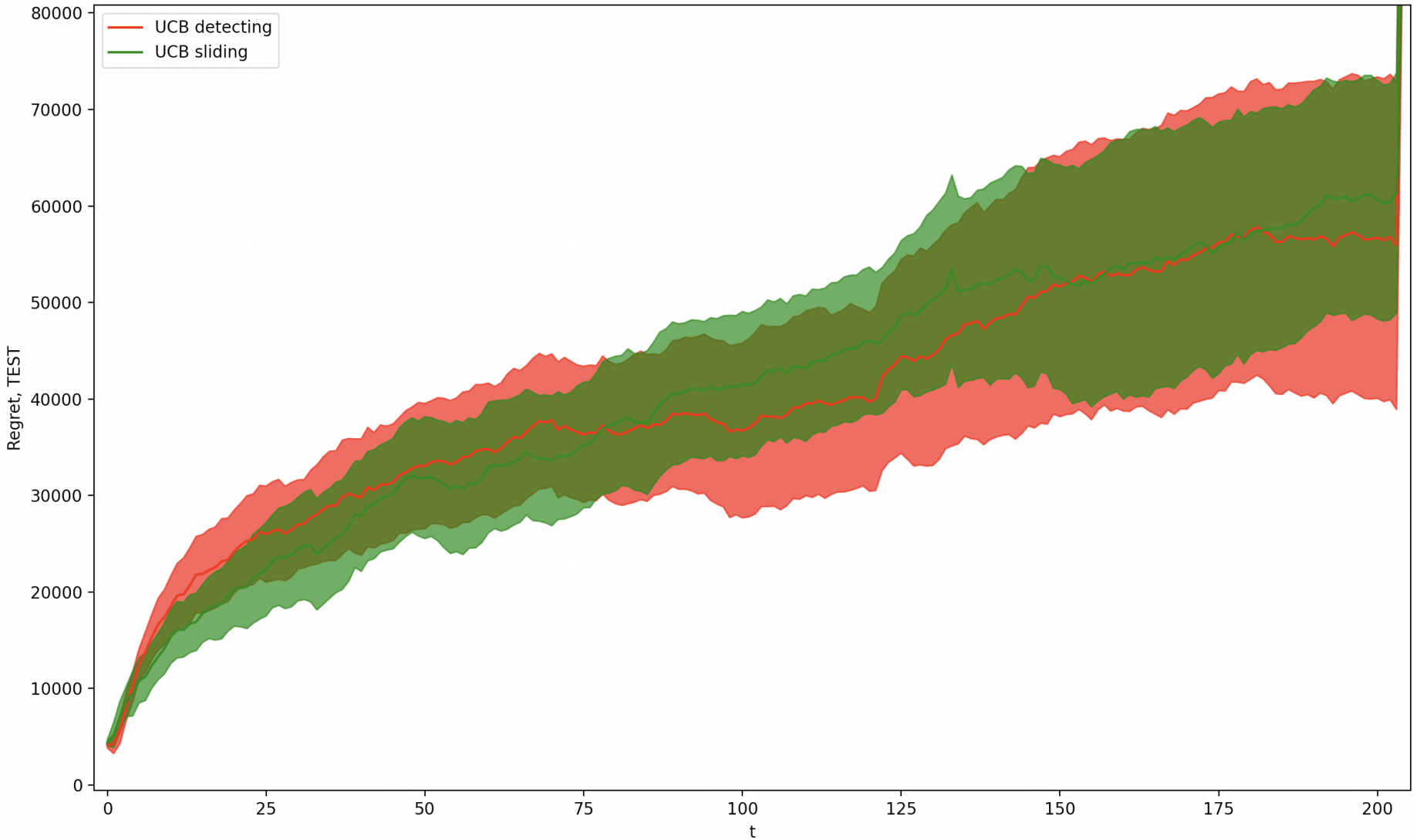
$$\text{NSolve[Distribution[}P\text{]==0.99 \&\& (0<=}P\text{<=1000)},P\text{]}$$

$$\{\{P \to 3.88411\}\}$$

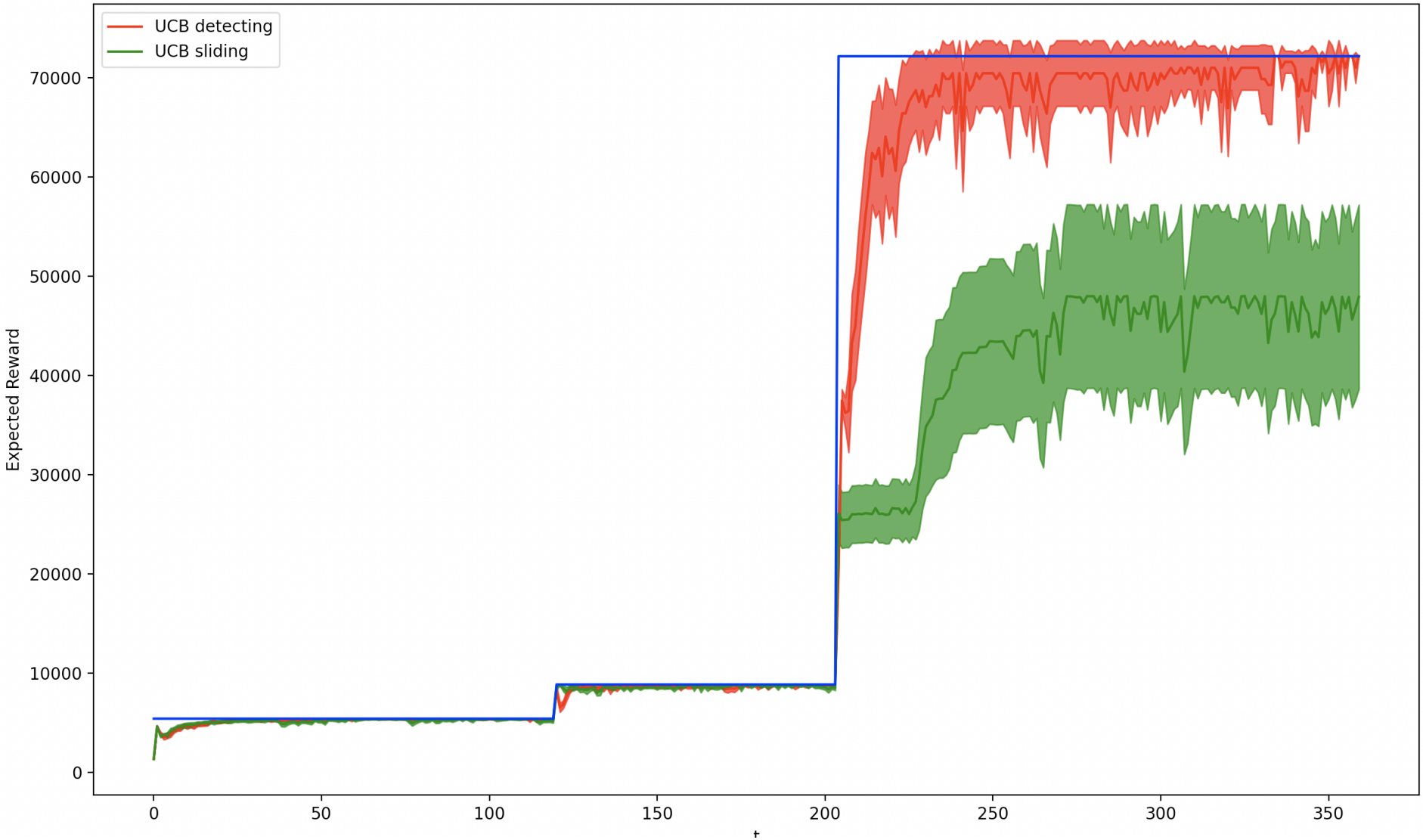
Regret Result Step 6



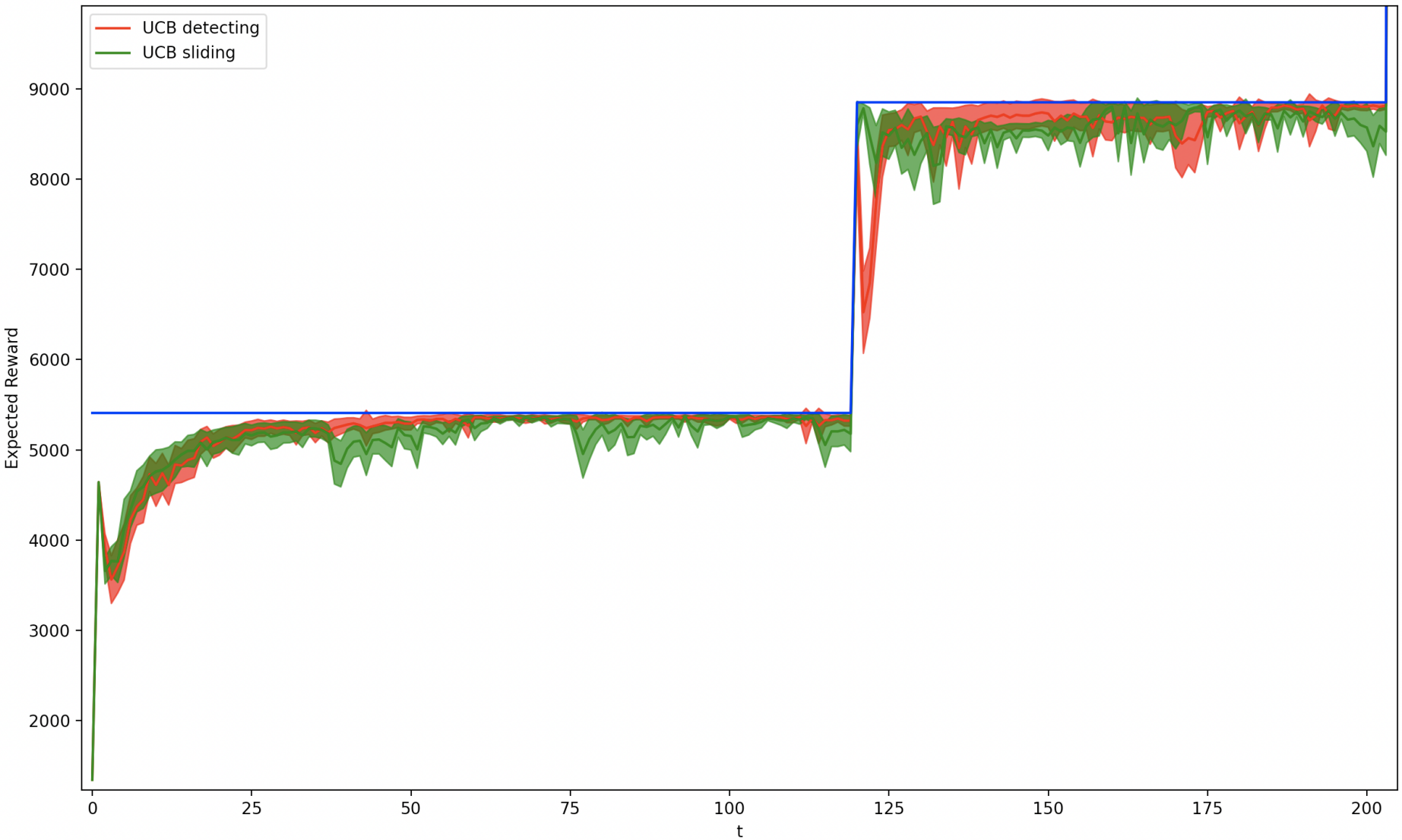
Regret Result Step 6: focus on first two phases



Expected Reward Result Step 6



Expected Reward Result Step 6: Focus on first two phases



Analysis and Comparison of the results

The environment has 3 different phases with breakpoints in 120 and 204. The first two phases represents a easier task while the last one an harder one, this can be deduced by the fact that while on the first phases both the learners reach the optimum, on the last phase only the more precise and powerful Detect Learner is able to achieve the optimum. So it seems that the window of the sliding learner is too short to achieve the optimal superarm. This is more clear in the chart of Slide 66 where it is shown the ***Expected*** reward w.r.t the optimal one, from the graph we notice how the sliding window is unable to achieve the optimum while the detect can. The Expected reward has been used instead of the one from the random environment in order to get a more polished view.

Step7: Context Generation

Context Generation

If the features characterizing the single classes are known the information can be leveraged to discriminate among user groups.
We assume that customers are characterized by two binary features that may be observed by a website:

- the first one is the geographical location of the user,
- the second one is the kind of user, a private customer or a company.

Companies are assumed to behave the same, i.e. have the same alpha-functions, independently from the location. So, even though features identify 4 different classes which generate the data, only three different profiles are present and should be targeted. The context generation algorithm should identify contexts and partition the user classes accordingly.

Contextual learning

After the first month, when usually the aggregated learners have stably reached the optimal superarm, every 10 days the ContextGeneration algorithm is run.
To simplify the computation, every split in the features is replicated for each product, thus the number of campaigns is given by $N_p * N_g$, where N_p is the number of products and N_g is the number of identified user groups.
Each campaign-daily budget couple is associated with a lower bound on the profit (neglecting the advertising budget) so that to have a conservative estimate of each context structure.

Context Manager

A **context structure** is a partition of the space of the features F, so a set of contexts k such that $\cap_i k_i = \emptyset, _i\, k_i = F$. Each context is a set of user classes as identified by thei features.
The aim of the Context Manager is to identify the different contexts starting from the data collected and assign to each of the contexts a set of learners to approximate the respective alpha-functions.
In our implementation, the Context Manager receives the data generated from each class and aggregates it to provide it to the respective learners. Every 10 days the context generation algorithm is run from scratch to identify a new context structure. The value matrices provided by the learners are aggregated into one and the Dynamic Programming algorithm is applied to the whole, providing a different budget for every context and product.

Context Generation

The Context Generation algorithm exploits lower bounds on the value of each campaign-budget couple to estimate the value of the optimal arm for each partition. Lower bounds are chosen to provide a conservative estimate of the provided value. Each subcampaign-budget couple is assigned a lower bound as $\underline{\alpha}_c(b) * N_c * v_c$, where $\underline{\alpha}_c(b) = \text{avg}(\alpha_c(b)) - \sqrt{-\frac{\log \delta}{2|D|}}$ is the Hoeffding lower bound for $\alpha_c(b)$ when $|D|$ samples are observed, N_c is the number of users in the context and $\delta = 0.95$ is the confidence on the bound.
The lower bounds are used to fill the value-matrix for the budget allocation algorithm which provides the optimal superarm and its value for the given partition. It is run recursively: a split is performed only if it provides an higher value than the current (w.r.t. the algorithm) context and this is repeated for every branch of the tree as long as some features are available.

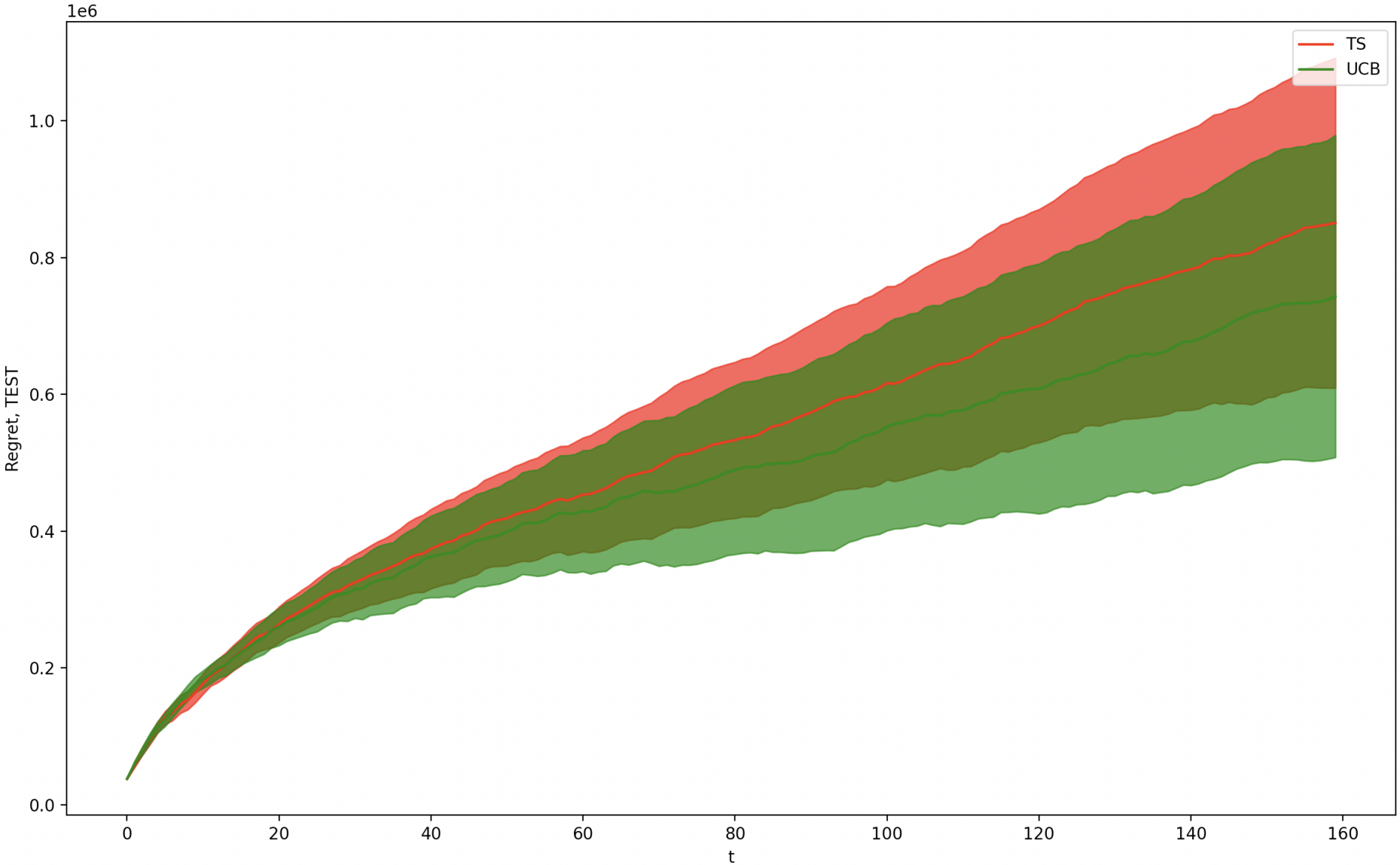
Edge weight estimation

Algorithm 6 Context generation algorithm

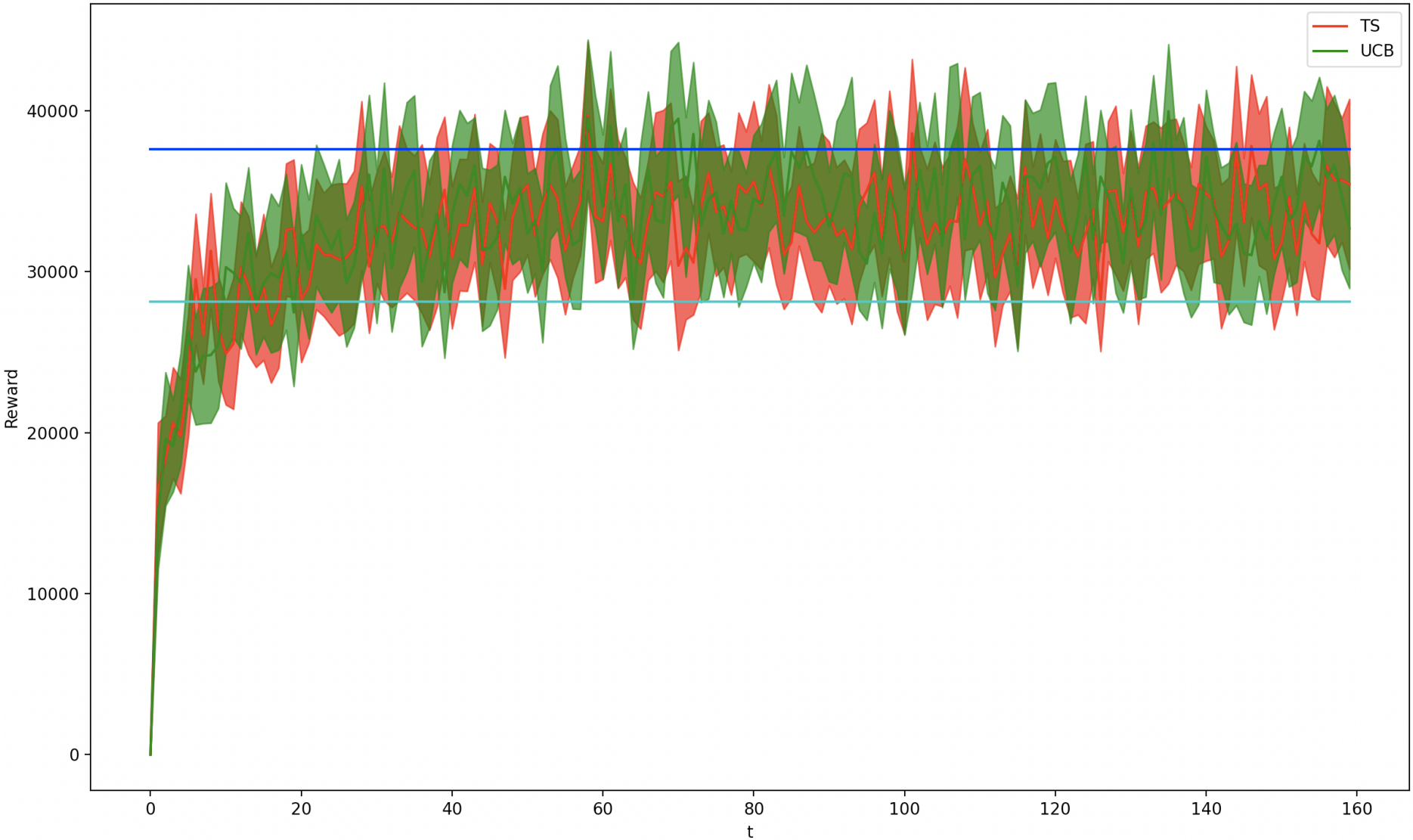
Require: data, a list of user classes with the corresponding realizations
Require: partition, a vector representing the partition into contexts of the user classes

values = $\emptyset$
no_split_value = evaluate_split(data, partition)
for feature in Features **do**
    partition = partition.split_on(feature)
    values.add(evaluate_split(data, partition))
end for
if max(values)<no_split **then**
    return partition
else
    Split on argmax(values), get left split (feature=0) and right split(feature=1)
    Take maximum and partition between left and right split
end if
return partition

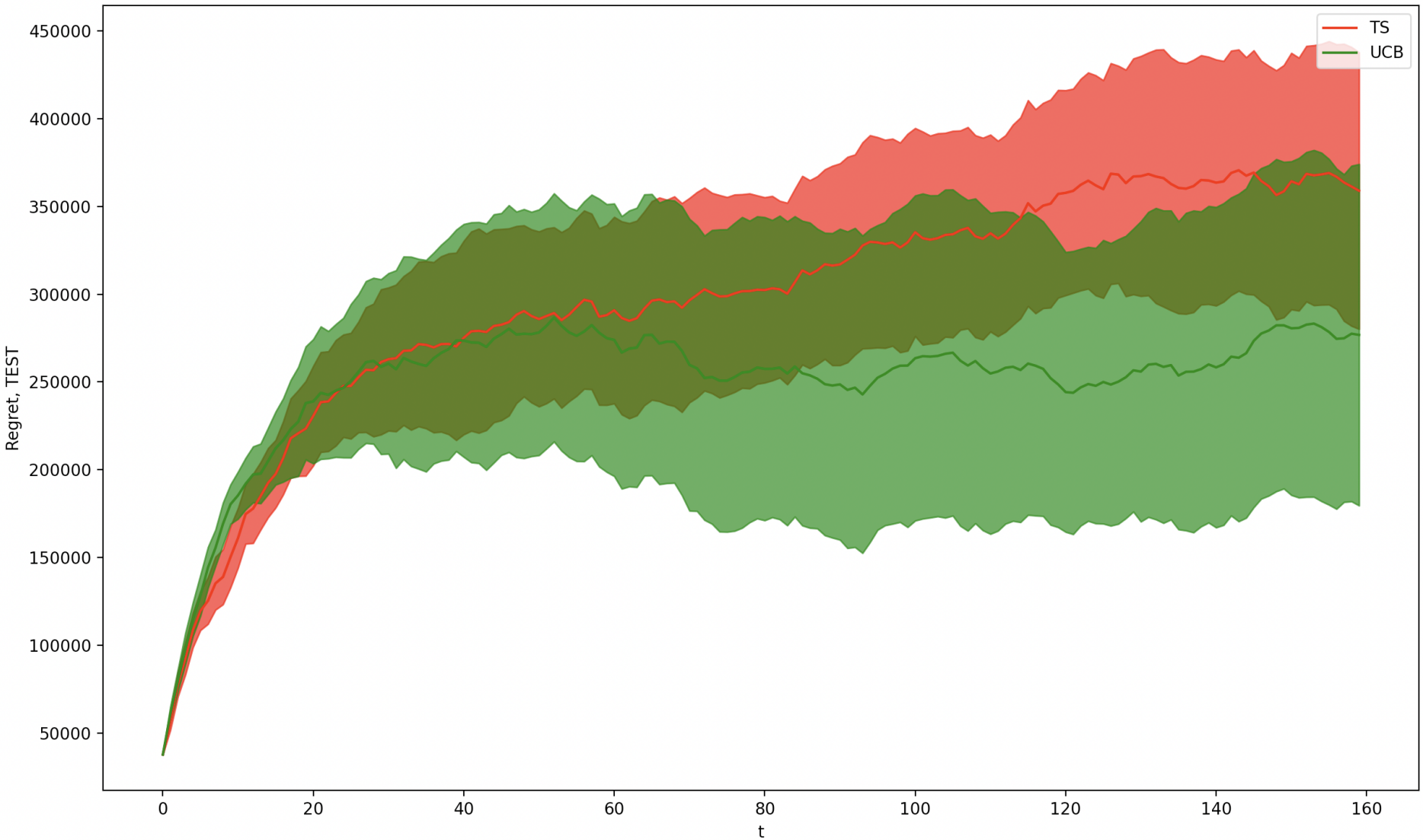
Regret Result Step 7



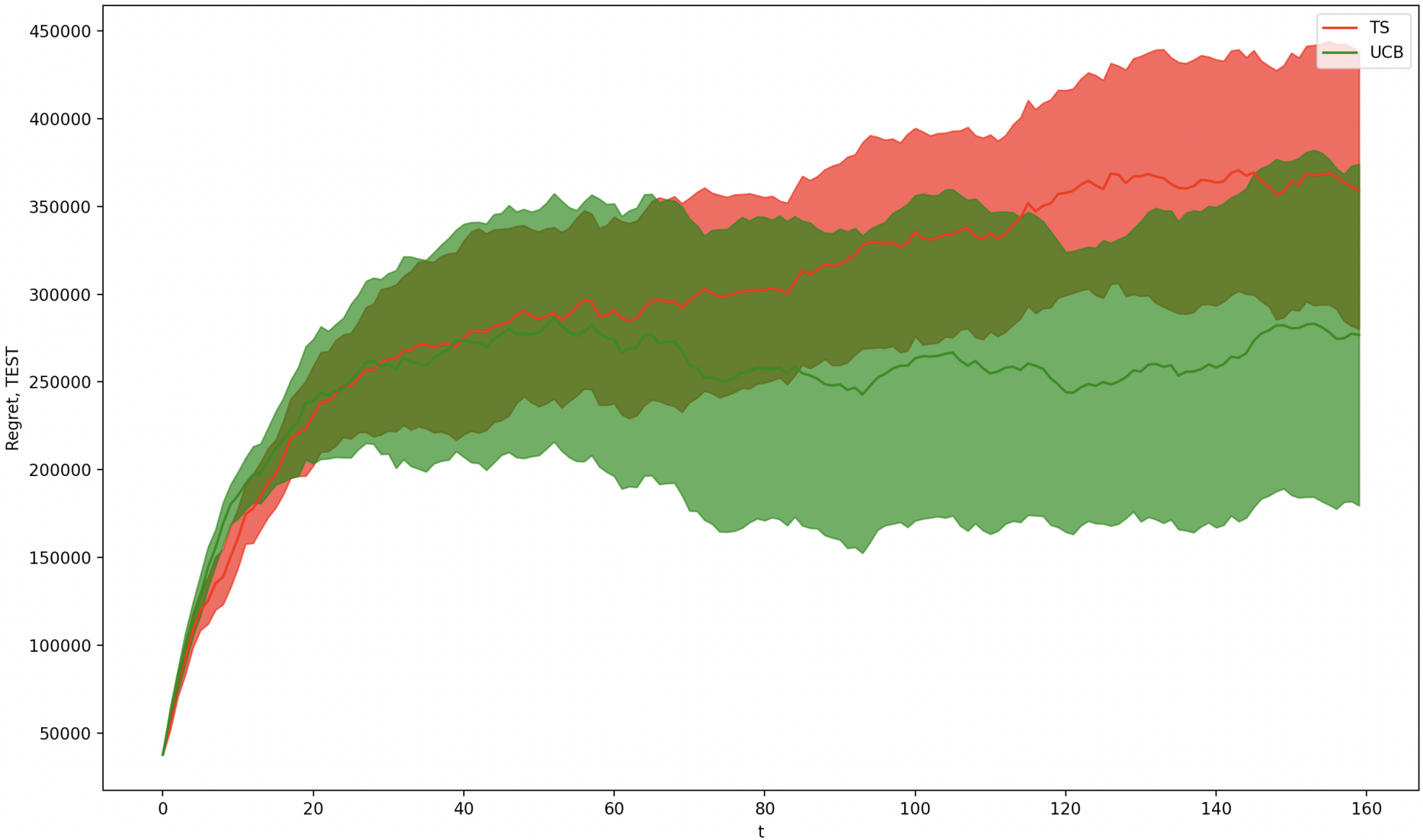
Reward Result Step 7



Regret Result Step 7: Best 18 experiments



Reward Result Step 7: Best 18 experiments



Analysis of the results

The task of finding the optimal contexts is an hard one and not in all experiments has been found such an optima, moreover it may happen that the learner gets stuck in a local maxima causing the regret to get extremely higher, this has happened with some of the experiment and they pushed the mean of the regret higher. The best 18 experiments are also shown in order to demonstrate that indeed the algorithm is able to achieve the optimum.
In the Reward plots the cyan line represent the optimum in the case of no context generation. It is a satisfying result the fact that even if not always the optimum is found, the algorithm is able to get a better result than the optima without contexts.

Non Fully Connected Graph

New Graph Considered

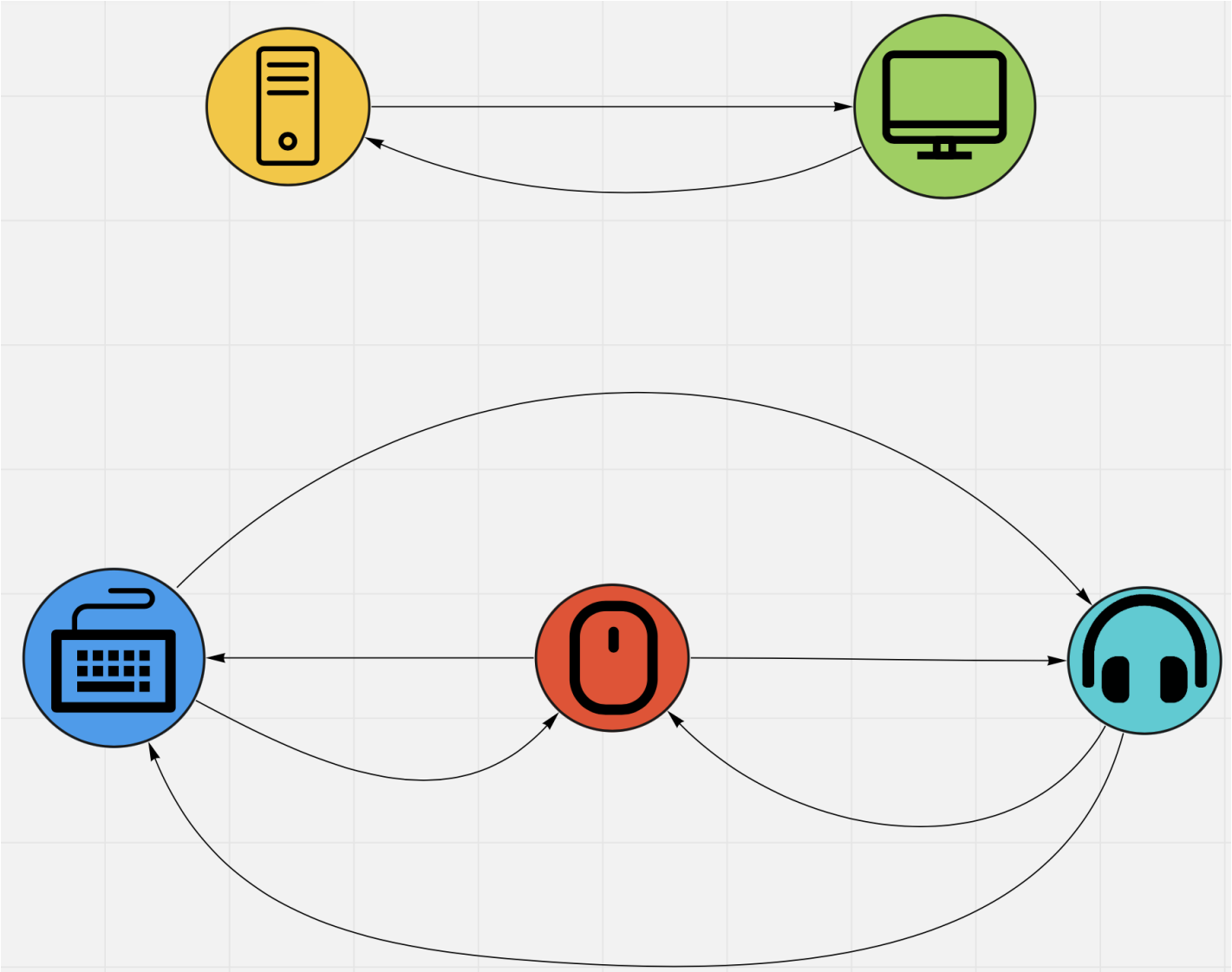
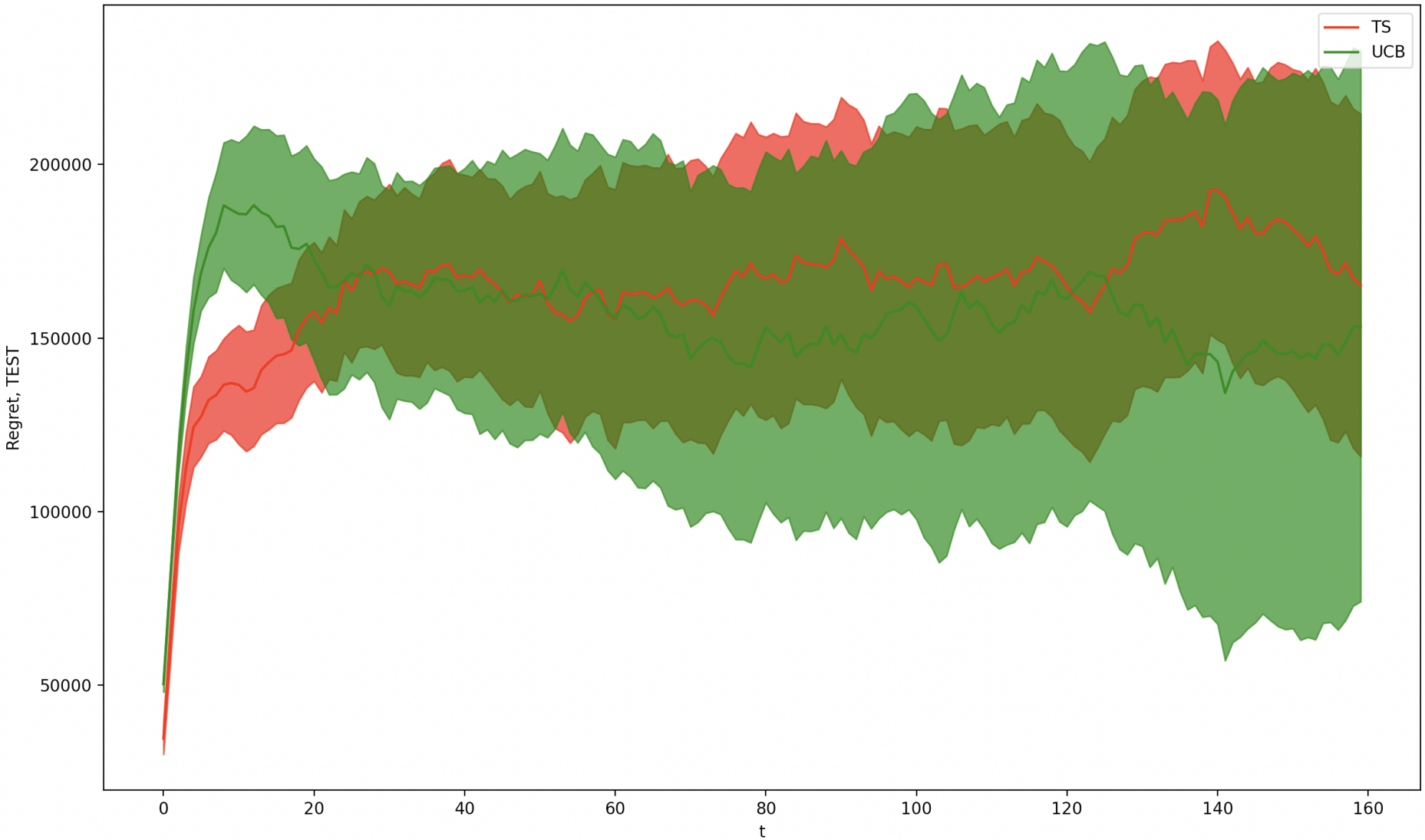


Figure 15: Graph with Tier1 and Tier2 products not connected

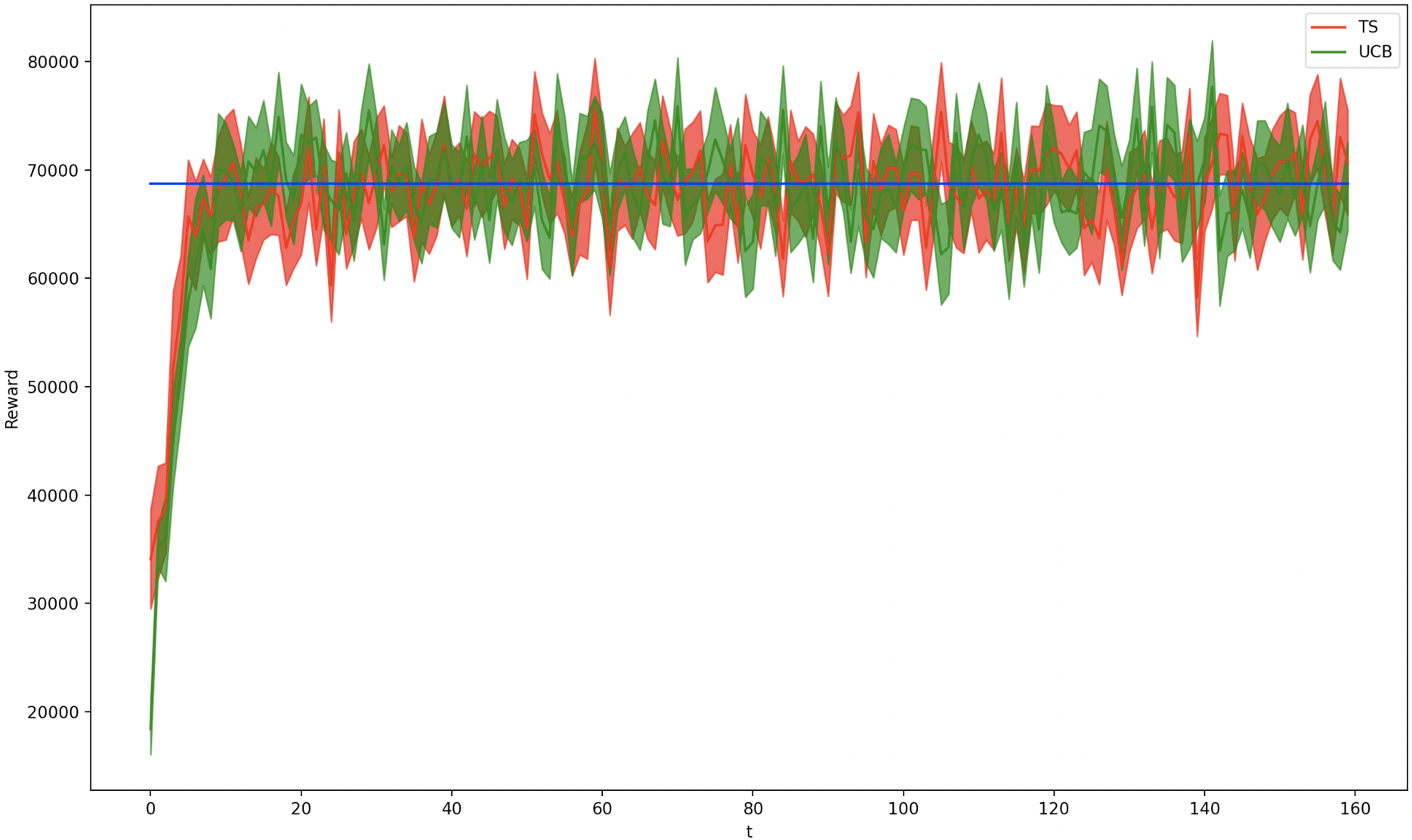
New Scenario Results

Below are listed all the results obtained in the case of not fully connected graph.
The general comments are the same as above so they are not repeated.
Also these results are satisfying and hence validate further the functioning of the algorithms implemented.

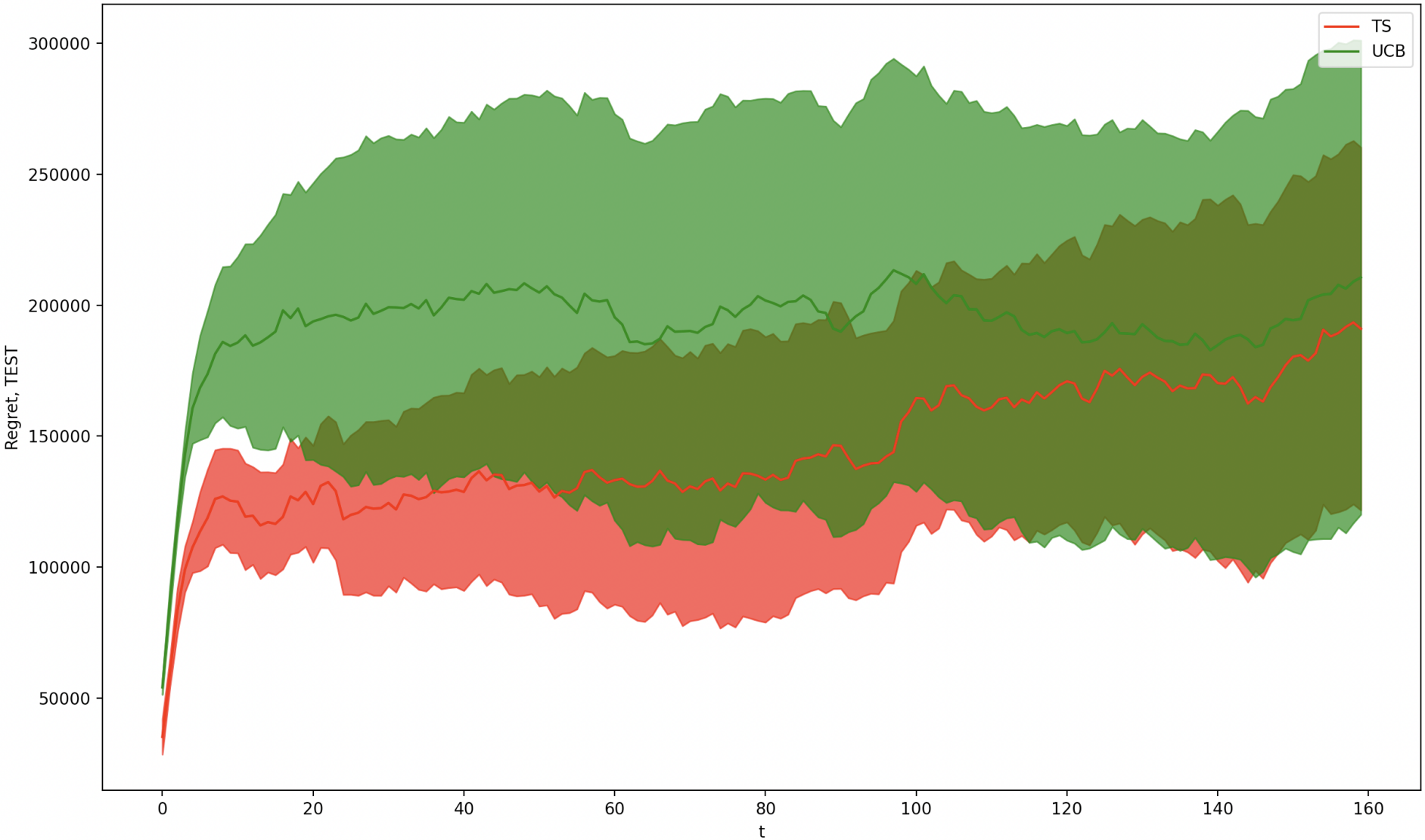
Regret Result Step 3



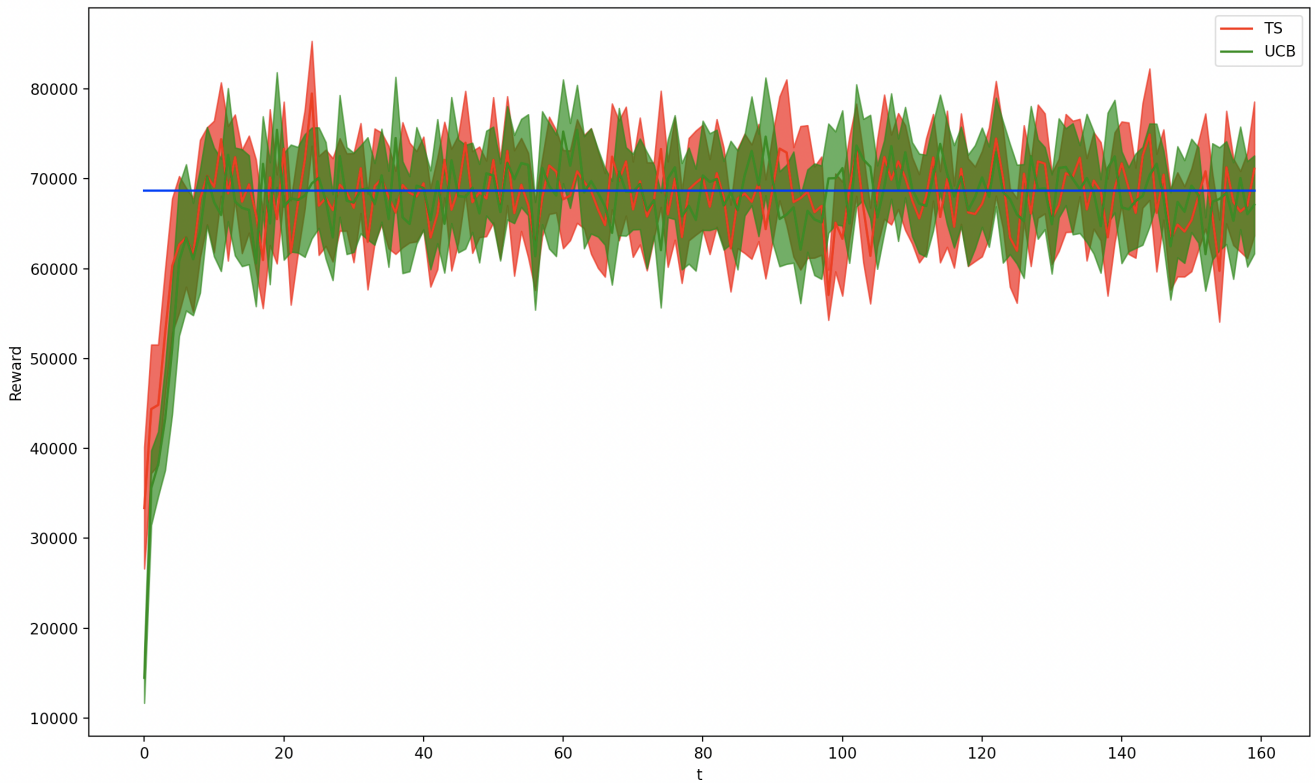
Reward Result Step 3



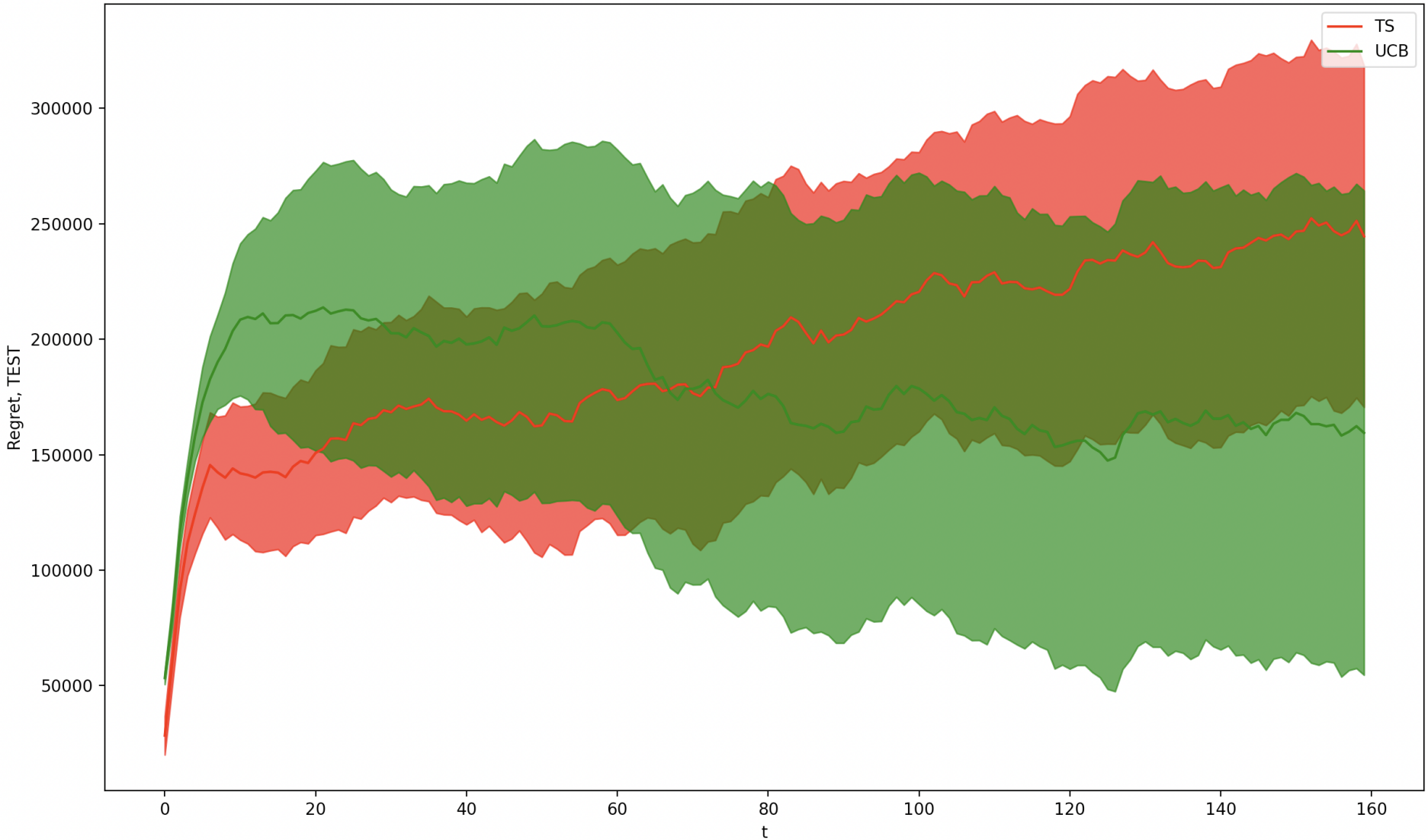
Regret Result Step 4



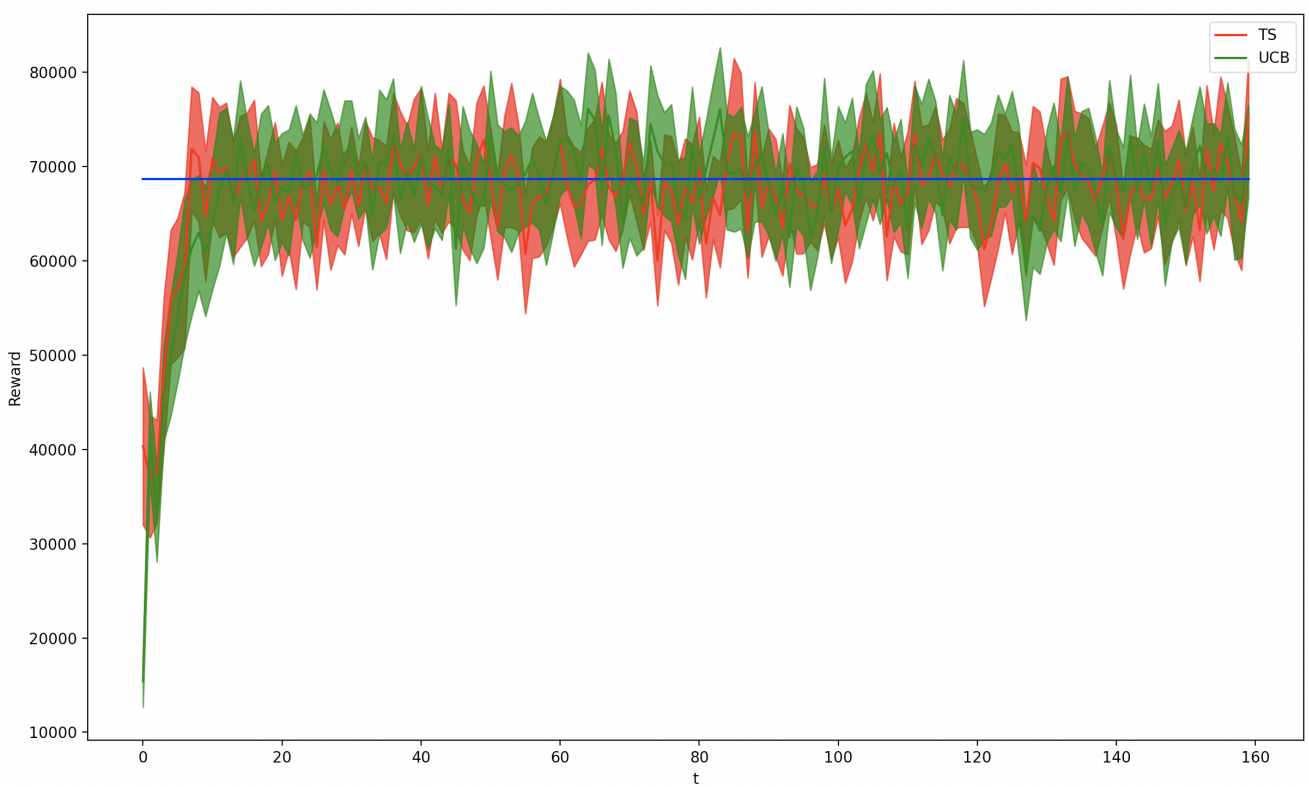
Reward Result Step 4



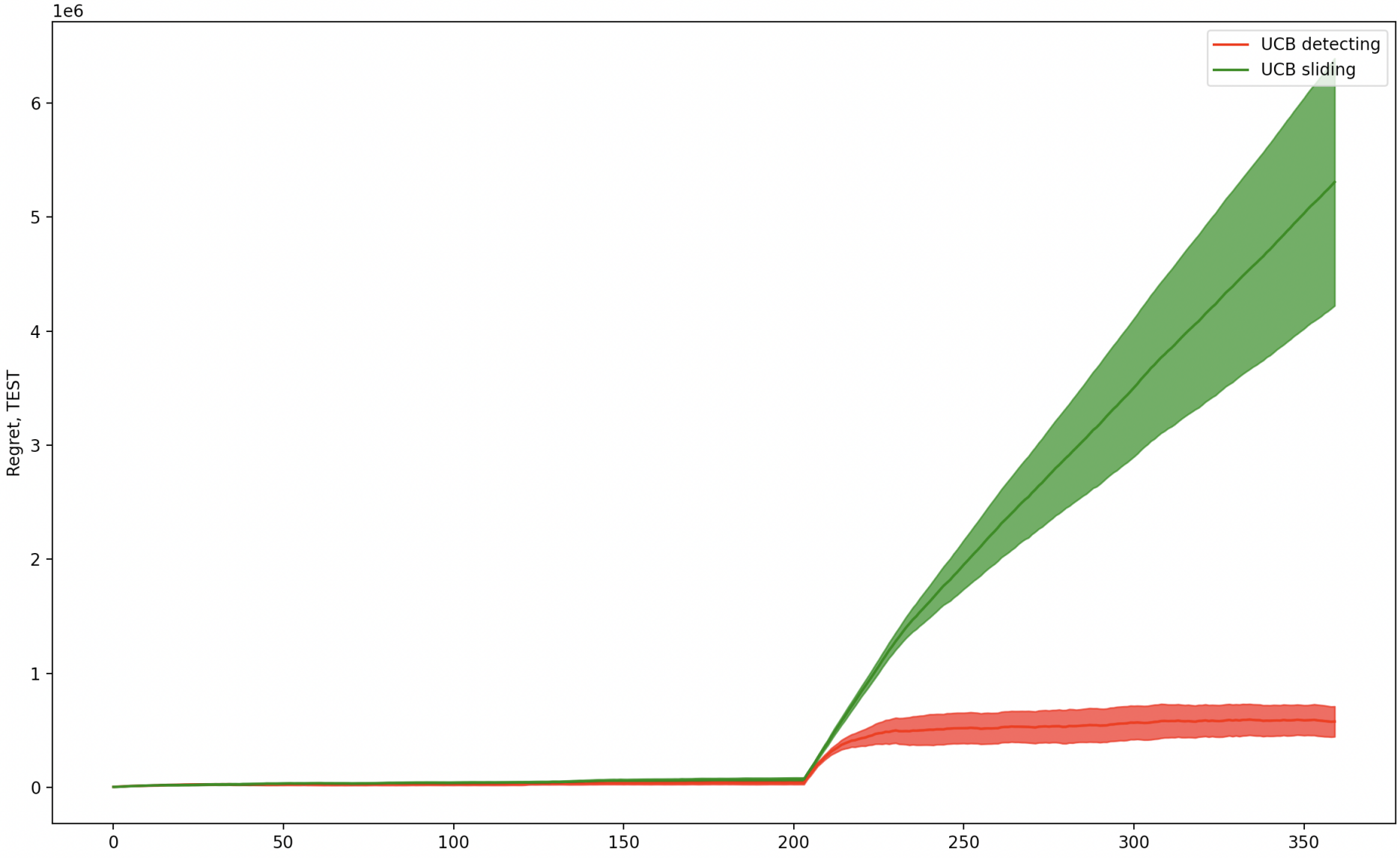
Regret Result Step 5



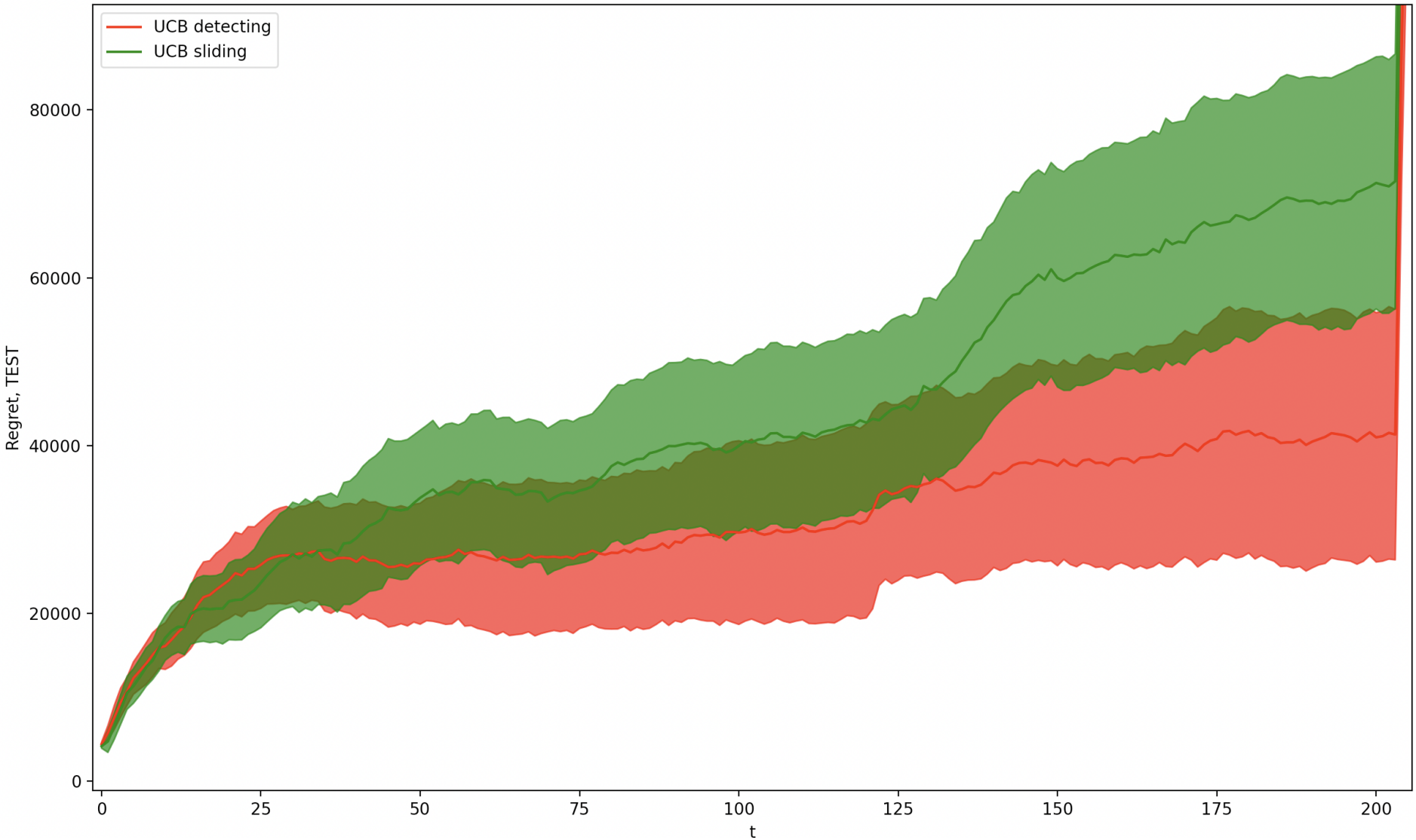
Reward Result Step 5



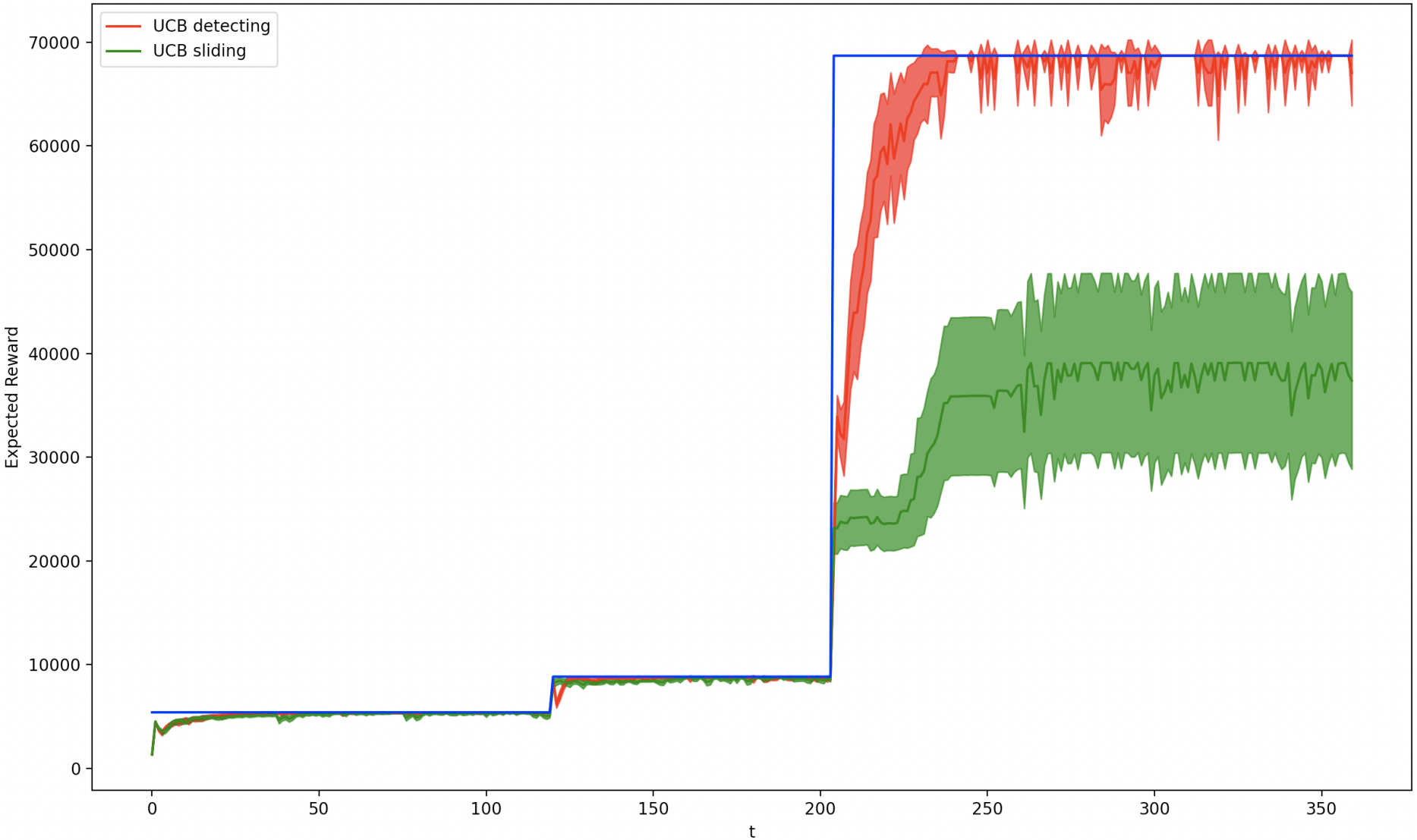
Regret Result Step 6



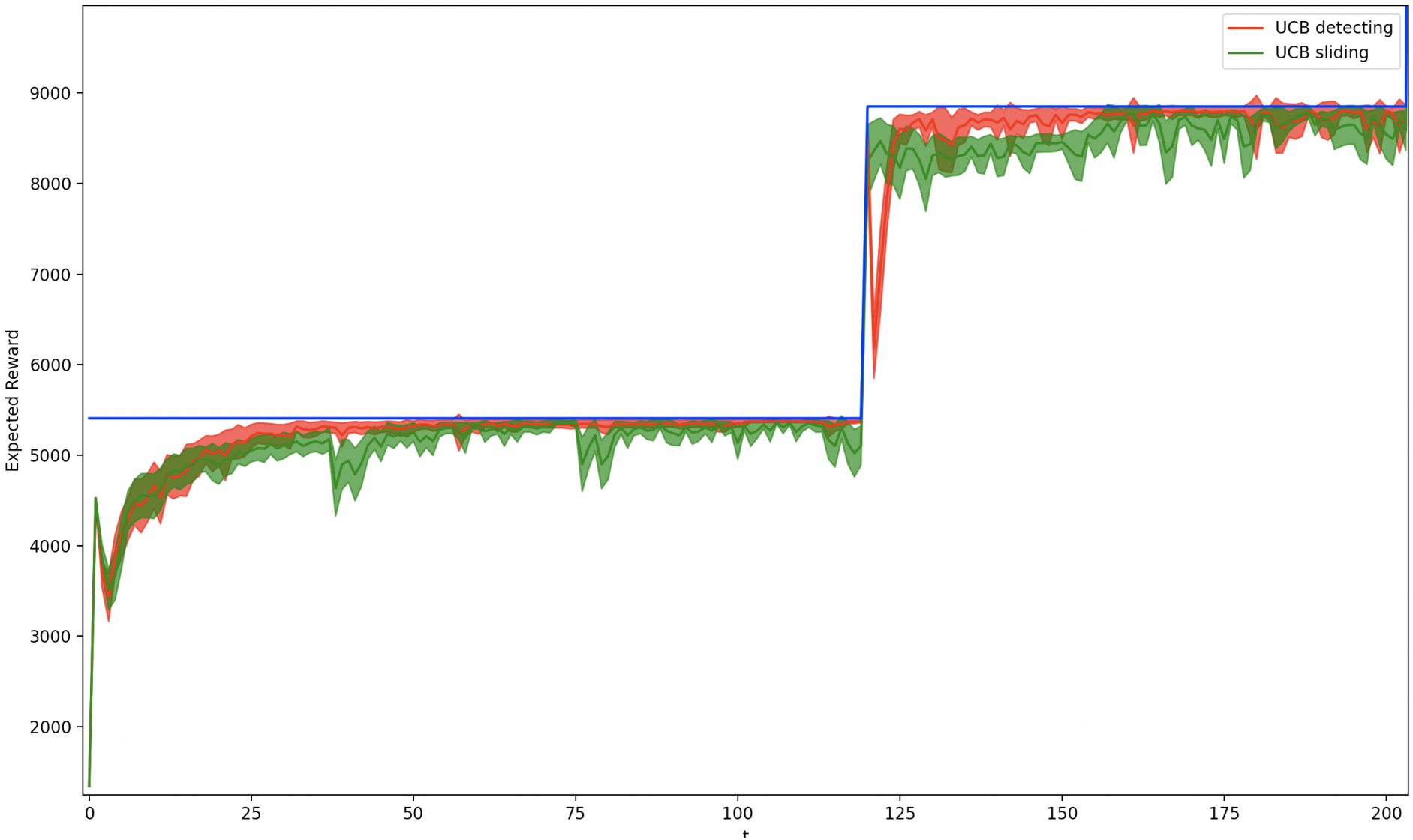
Regret Result Step 6: focus on first two phases



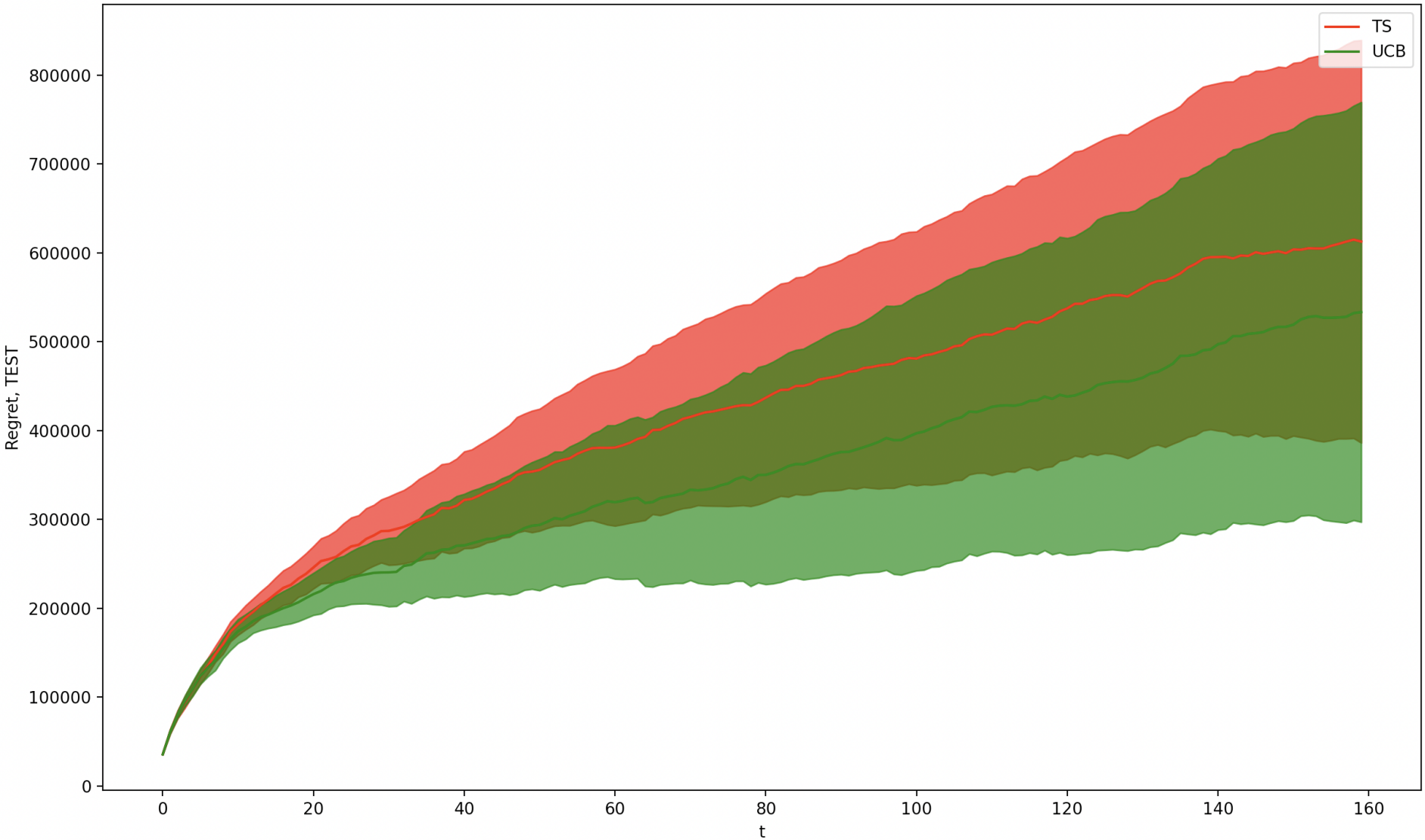
Expected Reward Result Step 6



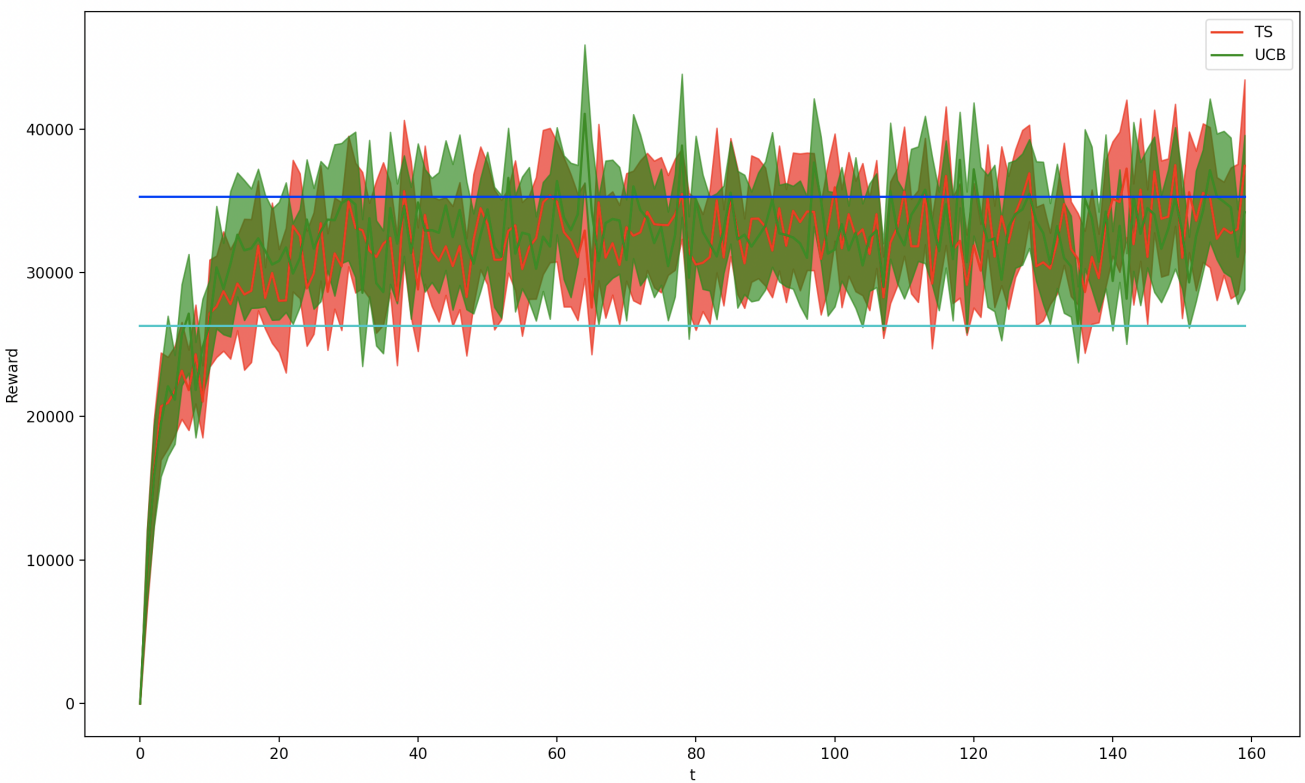
Expected Reward Result Step 6: Focus on first two phases



Regret Result Step 7



Reward Result Step 7



References

- Niranjan Srinivas, Andreas Krause,Sham M. Kakade,Matthias W. Seeger, Gaussian Process Bandits without Regret: An Experimental Design Approach, ICML, 2009
- Accabi, G. M., Trovo, F., Nuara, A., Gatti, N., Restelli, M. (2018). When Gaussian processes meet combinatorial bandits: GCB. In 14th European Workshop on Reinforcement Learning (pp. 1-11)