Dynamic allocation optimization in A/B tests using classification-based preprocessing

Emmanuelle Claeys, Pierre Gancarski, Myriam Maumy-Bertrand, Hubert Wassner

Dynamic allocation optimization in A/B-Tests using classification-based preprocessing

Emmanuelle Claeys,[1], Pierre Gançarski[2], Myriam Maumy-Bertrand[3], and Hubert Wassner[4]

[1] University of Toulouse, I.R.I.T. laboratory
emmanuelle.claeys@irit.fr
www.emmanuelle-claeys.com
[2] University of Strasbourg, ICUBE laboratory, Strasbourg, France
[3] University of Technology of Troyes, Troyes, France
[4] AB Tasty, Paris, France

Abstract. An A/B-Test evaluates the impact of a new technology by running it in a real production environment and testing its performance on a set of items. Recent development efforts around A/B-Tests revolve around dynamic allocation. They allow for quicker determination of the best variation (A or B), thus saving money for the user. However, dynamic allocation by traditional methods requires certain assumptions, which are not always valid in reality. This is often due to the fact that the populations being tested are not homogeneous. This article reports on a new reinforcement learning methodology which has been deployed by the commercial A/B-Test platform AB Tasty. We provide a new method that not only builds homogeneous groups of users, but also allows the best variation for these groups to be found in a short period of time. This article provides numerical results on AB Tasty's data, in addition to public datasets, tha demonstrate an improvement over traditional methods.

Keywords: A/B-Test, Bandit strategies, UCB strategies, Conditional inference tree, Non linear bandit, Regret minimisation.

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2021.3076025

1 Introduction

In a number of economic, industrial or even social fields, it may be interesting to evaluate the relevance of modifications to an existing entity according to one or more objectives by directly and concretely comparing the different variations resulting from the modifications. For instance, an e-marketing team can look for the best modification to apply to a given web page to increase sales. A medical laboratory may want to find the best drug composition modification to

save more patients. A company may want to define the best modification to an industrial process to increase product quality. Such a task requires a mechanism to evaluate each variation in order to make the optimal choice according to a defined objective in the given context. *A/B-Test* based approaches have been proposed to respond to this problem [1] and have recently generated renewed interest, particularly from their use in e-marketing. An *A/B-Test* consists in assigning the *items* (patients, visitors, goods to be produced, ...) to the different variations in order to evaluate the relative performance of each item. During this *exploration phase*, it is assumed that the result, called *reward*, of each *assignment* can be observed and used by the algorithm to evaluate the performance of each variation. At the end of this exploration phase, the *user* can better decide which variation will be used in the future (i.e., in *production phase*) according to their relative performance.

An important characteristic of such methods is that the decision to assign an item to a variation is irrevocable. For instance, for the duration of the test, a visitor will always see the same web page on each of his/her visit, regardless of the number of visits. It is impossible to know what he/she would have done if he/she had been assigned another variation. Consequently, the population that has been assigned to a variation is distinct from those assigned to any other. Finally, it is assumed that items are unaware of their participation in a test and thus of the existence of different variations.

A classical approach to the exploration phase is referred to as the frequential approach and consists in assigning items to the different variations according to explicit fixed ratios (*static allocation*) for a given period of time which is unfortunately difficult to define *a priori*. Experiments have shown that a user tends to overestimate it, causing an inferior variation to have a large detrimental effect on the result for a long period of time. In this case, the obtained cumulative reward will be much lower than that which would have been produced by the allocation of each item to the optimal variation. This difference, called *regret*, increases with negative impact. Reducing the exploration phase may reduce regret, but may also lead to a lack of data needed to calculate performance. Therefore, in addition to determining the best option, the challenge of A/B-Test methods is to also minimise regret. Nevertheless, it is important to note that regret cannot be calculated during the observation phase as the optimal variation is obviously unknown *a prior*: the objective of the test is, by definition, to determine it. Finally, in most cases the sooner the algorithm finds the solution (i.e. the sooner exploration can be stopped), the better.

To address this problem, new A/B-Test methods perform *dynamic allocation* of items based on bandit algorithms consisting in adapting the allocation of visitors according to the obtained rewards and thus gradually tipping the visitors towards the optimal variation. This dynamic allocation is usually achieved using the probabilistic comparison criteria of the empirical reward distribution of each variation. The idea is to maintain and update the gain estimate of each variation and to allocate items accordingly. It is therefore a matter of favoring the most

promising variation while continuing to refine the remaining gain estimates by continuing to allocate items to potentially sub-optimal variations.

Experiments and theoretical studies have shown that dynamic allocation [2] provides better results in terms of cumulative regret, and is faster at determining the best variation. Therefore, numerous methods implementing dynamic allocation based on bandit algorithms have been proposed [3,4,5] and have proved their ability to find optimal variations in the general case. Nevertheless, experiments also show that these methods often fail when the reward obtained by an item depends on both the variation and the item itself [6]. For instance, in web marketing, visitors naturally tend to click and buy differently according to their own financial resources or their geographical location. In medical treatment, the efficiency of a drug often depends on the age and/or gender of the patient. To address this problem, bandit algorithms have been extended to form *contextual bandits*, which take into account each visitor's *context*, i.e. its characteristics (age, origin, gender, etc.) when allocating it in order to perform more relevant allocations. Methods such as KernelUCB [7] and LinUCB [8] (Section 3.2) have demonstrated not only the benefits of such an approach, but also their limitations (which strongly reduce their practical use), including in particular:

- large latency (corresponding to the time required by the algorithm to allocate an item to a variation),
- the need for a large number of items before finding the optimal variation,
- a lack of explainability of the assignments made by the algorithm

However, it is obvious that in many cases items belong to natural groups (social classes, levels of study, age classes, ...) for which the reward distribution each variation can differ. For instance, for a given web page students may behave differently from workers or retired people and, in fact, can be differently impacted by a modification. Unfortunately, these groups are often very difficult to determine because they strongly depend on the application domain and on the test itself.

In this paper we propose an original A/B-Test method called Ctree-Ucb which, instead of using a contextual bandit, is based on the use of several non-contextual bandits, each dedicated to a particular group of items. Our proposal consists in automatically creating homogeneous groups in a pre-processing step using a conditional inference method using information (obtained rewards, item characteristics, temporal information, etc.) derived from items subjected to an existing variation in production phase before the test. Then, in the exploration phase, a non-contextual bandit is dedicated to a group and is used to find the optimal variation associated with the group. To achieve this, each new item submitted to the A/B-Test is classified into a group before being transmitted to the associated bandit for its allocation to a variation.

The remainder of this paper is organized as follows. Section 2 presents the bandit model with an illustrative example. Based on this example, Section 3 gives a comprehensive literature review of existing approaches and focuses on contextual strategies able to take into account the characteristics of the items. Section 4 details the proposed methods. Sections 5, 6 and 7 analyse and discuss

the results obtained with this method on real data provided by AB Tasty[5]. Finally the conclusions of the study are drawn in Section 8.

For readability purposes, the remainder of this article focuses on tests with only two variations. All our propositions are however directly and easily extended to tests with more variations.

2 Bandit problem

2.1 The multi-armed bandit model

The first definition of the multi-armed bandit model was introduced by Lai and Robbins [9], by an analogy with casino slot machines. For a player, it is a matter of choosing from a machine with several arms the one presenting, for him, the best expectation of gain. To do that, each time the player plays an arm and obtains (or not) the gain, he/she updates the gain estimates of the arm. The player's goal is to find the best arm, called the *optimal arm*, while limiting the number of tries. As introduced in Section 1, bandit-based approaches are frequently chosen to concretely implement dynamic allocations: at each iteration t, corresponding to the arrival of an item c_t, the bandit algorithm chooses an arm a in the set of possible arms $\mathcal{A}$ according to its own strategy π. Then, the reward $X_{c_t,a=A_t}$ obtained by the assignment of the item to the chosen arm a is observed. The main characteristic of strategy π is that the allocation of the items depends on the reward expectation of the variations.

2.2 The bandit paradigm as reinforcement learning

The first mention of the bandit problem appears in [1]. This paper presents a reinforcement learning problem where an autonomous agent must learn the actions to be taken from experience in order to optimise a quantitative reward over time (per the definition of reinforcement learning: "agents ought to take actions in an environment in order to maximize some notion of cumulative reward" [10]). The agent evolves in an environment and makes decisions based on its current state. In return, the environment provides a reward, which can be positive or negative. The agent seeks an optimal decision-making behaviour (called *strategy* or *policy*, which is a function associating the action to be performed with the current state) through iterative experiments in the sense that it maximizes the sum of rewards over time. Moreover, the agent must find a balance between exploration of uncharted territory and exploitation of current knowledge.

Indeed, an A/B-Test (and more specifically the bandit algorithm) can be seen as an agent with partial knowledge of the world (the different variations, the items having been subjected to these variations, and the rewards obtained so far). Knowledge of this world is very sparse at the beginning of the test but is reinforced by observing the rewards of each variation in accordance with the context of each item. Thus, initially, the bandit does not know anything

[5] `https://www.abtasty.com`

about the distribution of the rewards of each arm. It has to *explore* to find it by assigning items to the different arms to learn these distributions with the risk of earning less reward. However, at the same time, it has to *exploit* by assigning the arm which it estimates to be the most rewarding with the risk of not discovering the optimal arm. This well-known exploration-exploitation dilemma has been extensively studied through the multi-armed bandit problem in [11].

Finally, when the agent has identified the best variation according to the characteristics of the visitors, the user can:

- put one of the variations (A or B) into production,
- compose another variation to be tested.

2.3 Cumulative regret

Let $\mathcal{A}$ be the set of possible arms (with $|\mathcal{A}| \in \mathbb{N}^+$), a^* the optimal arm and $X_{A_t,t}$ the reward obtained at iteration t by the item with arm A_t selected. The *simple regret* r_t is defined by $(X_{a^*,t} - X_{A_t,t})$ where $X_{a^*,t}$ is the reward the item would have been obtained with a^*. Then, the *cumulative regret* R_T is defined as the sum of simple regret r_t over all the T items. This cumulative regret and its evolution during the tests is the main criterion used to evaluate efficiency of bandit-based algorithms. From the definition of A/B-Test, a^* is unknown *a priori*. To evaluate R_T during a test, and follow its evolution, the value of r_t is estimated by $r_t = \max_{a \in \mathcal{A}}[X_{a,t}] - X_{A_t,t}$. The cumulative regret after n iterations can thus be estimated by:

$$R_n = \sum_{t=1}^{n} \max_{a \in \mathcal{A}}[X_{a,t}] - \sum_{t=1}^{n} X_{A_t,t}, \tag{1}$$

3 State of the art

Three characteristics can discriminate the different strategies:

- All the strategies π are based on the strong hypothesis that the distribution of all arm rewards follows the same law (Bernoulli distribution with Thompson sampling [1,**?**], Gaussian distribution with Ucb [12]) or otherwise make no assumption.
- Two mechanisms assigning items can be defined. The first qualifies as *non-informative* as it uses no information about items (only rewards are used to make a choice), while the second qualifies as *contextual* as it considers item characteristics when assigning to a variation.
- Different mathematical models can be used in the choice mechanisms such that the best arm (based on previous observations) is chosen according to predefined probabilities (such as the Epsilon-Greedy algorithm [13,14]) or using adaptive probabilities (for example Softmax exploration [15]).

3.1 Non-informative strategy

A *non-informative strategy* assumes that the best arm is the same for all (or at least, for the majority of) items and therefor, the arm is allocated independently of the characteristics of the item.

Ucb strategy The UCB strategy is a non-informative method based on an optimistic Bayesian strategy (with probabilistic upper bounds of the real average). The principle of its strategy π to assign a new item to an arm, is to use an overestimation of the empirical average $\hat{\mu}_{a,t}$ for each arm a, the total number of items, and their allocation to different arms. Concretely, an arm is chosen if it is promising (because its estimated average is high) or/and seldom explored (see Algo. 1 where $T_a(t)$ is the number of times that arm a has been chosen[6]).

Algorithm 1 UCB algorithm

Require: $\alpha > 0$
Require: Assign at least one iteration to each arm a
1: **loop**
2: $c_t \leftarrow$ a new iteration
3: $A_t = \underset{a \in \mathcal{A}}{\text{argmax}}\{\hat{\mu}_{a,t} + \alpha\sqrt{\frac{2 * \log(t)}{T_a(t)}}\}$
4: Assign arm A_t to c_t
5: $X_{A_t,c_t} \leftarrow$ the arm A_t reward
6: Update $\hat{\mu}_{A_t}$ and $T_{A_t}(t)$
Output: A sequence of arm choices (A_t) and rewards X_{A_t,c_t}

In fact, the $\hat{\mu}_{A_t}$ estimators may not be relevant at the beginning of the test, due to the small number of items considered [16]. To get around this difficulty, [12] proposes to calculate an overestimation of this average, called the *upper confidence bound*. The authors justify their proposition by a policy known as "optimistic in the face of uncertainty" and demonstrate good results.

This upper bound is the sum of the empirical average of the reward obtained so far in addition to an exploration bonus (also known as the confidence interval). It depends on the number of items assigned and observed. The more observations an arm makes, the more the arm bonus decreases. If ν_a is Gaussian random for all a, this bound will always be higher than the real average. Thus, the authors define the upper bounds of each arm by:

$$Upper_{\text{UCB}}(a,t) = \alpha\sqrt{\frac{2 * \log(t)}{T_a(t)}}, \tag{2}$$

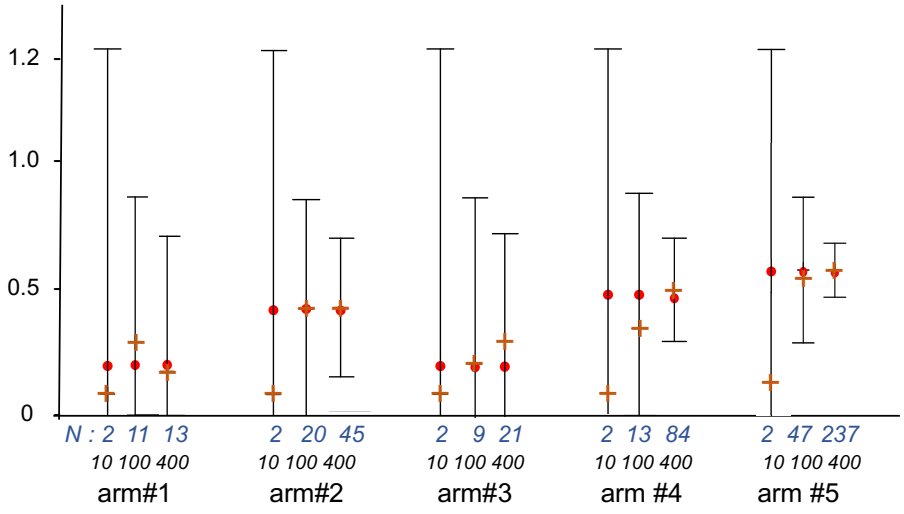
[6] Note that α is different from the α risk commonly used in statistics.

where α is a positive real parameter given by the user. In the initial version of Ucb, $\alpha = 1$ but in practice it has been shown that the optimal choice of this value depends on the arm distributions [17].

The π algorithm consists in choosing the arm with the highest upper bound. After each assignment, $\hat{\mu}_{A_t}$ is updated and its bound is reduced (Equation (2)).

Therefore the confidence interval depends on $T_a(t)$. So the higher $T_a(t)$ (i.e.,the more the arm a is chosen), the lower the overestimation. As the overestimation of the chosen arm average decreases, the empirical average towards its real average. The upper bounds of the unchosen arms remain unchanged.

Figure 1 shows an example of the confidence bound evolution for five arms according to the number of tested items.



N : number of times the arm has been selected, • : real average

Fig. 1: Confidence bound evolution according to empirical average (+) after 10, 100 and 400 items.

Limit of the Ucb strategy The Ucb strategy is very efficient when the real distribution of rewards is Gaussian, an assumption that can be verified retrospectively using static allocation. Unfortunately, experiments have shown that this assumption is rarely valid. In fact, the upper bound is not reliable. Consequently, Ucb requires more items to find a^*[18,19]. Moreover, if the reward presents extreme values, the convergence can be very long. Nevertheless, it has been proved that the a^* will eventually be found. As such, despite these imperfections, since Ucb focuses on the average reward that leads to a low complexity

and since it is generally well understood by the user of an A/B-Test, overall it responds to the problems of interpretability and limited computation time.

The remaining problem is that the identification of a^* may not be the only objective of the A/B-Test. Rather than looking for the variation that maximises the gain on average, the A/B-Test user can instead look for the best variation according to different sub-populations (as quickly as possible).

Indeed, we suppose that there exist two sub-groups of items having different responses to the test. For instance, in the medical field, an alternative treatment may be efficient for the elderly and not for young people. In marketing, a page can be optimal only for smartphone users. In this case, ν_a is very far from a Gaussian distribution. Thus, recent approaches assume that ν_a is a mixture of Gaussians particularly in contextual-based strategies. The idea of the contextual strategy (presented below) is that the reward X_{A_t,c_t} depends on both the arm assigned and an item's characteristics (features).

3.2 Contextual strategy

Contextual approaches assume that there exists sub-groups of items, each presenting a different reward distribution. Nevertheless, experiments show that it can be difficult to define such groups. Asking the user to define them is often unproductive, as they rarely have a clear vision of these different groups. To overcome this problem, it is assumed that there are *a priori* unknown links between, on the one hand, the *context* of the items (i.e., the characteristic vectors describing the items [20]) and the groups, and on the other hand, the groups and the averages of the rewards obtained. To fit this link, two approaches can be considered:

- In *contextual bandit*, this link is modelled by a unique regression function, i.e. the groups and their associated average rewards are directly set by the bandit during the test (Section 3.2).
- In two step-based approaches, the groups are set using a pre-processing step of the A/B-Test (Section 3.2).

Contextual bandits In contextual bandits, rewards are assumed to be generated from an unknown function depending on the item features (characteristics) and the chosen arm. The objective is to fit this function during the test. Concretely, assumptions are made about the type of function, such as linearity. Strategies such as Lin-Ucb are based on this idea.

With linear regression based bandits, the arm parameters are often calculated by matrix inversion, which can be time consuming, depending upon the number of item features d [21]. These approaches have shown their theoretical and practical ability to reduce cumulative regrets [22]. In particular the Lin-Ucb algorithm is one of the most popular form of such strategies due to its performance and interpretability. Figure 2 shows the cumulative regret over t calculated with simulated data and a linear reward function. From this figure,

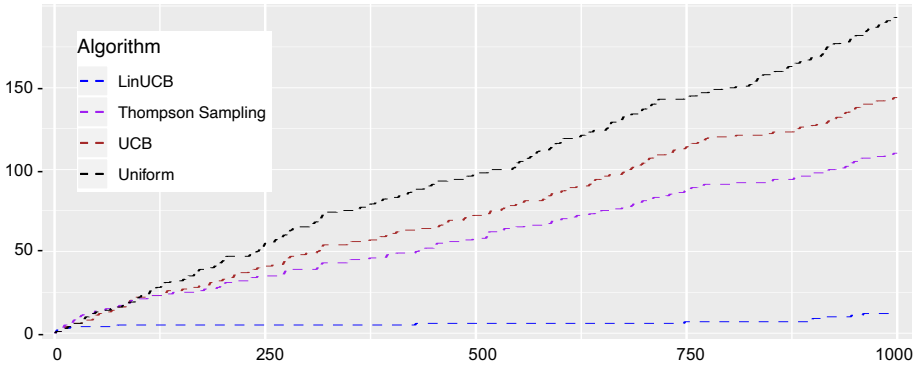


Fig. 2: Cumulative regret evolution according to number of simulated data.

one can see that the Lin-Ucb algorithm outperforms Thompson Sampling, random, and Ucb.

Statistical techniques have been proposed for cases in which the linearity assumption is not valid. In [23], the authors propose a strategy based on the Generalized Linear Model (GLM), called GLM-UCB. This strategy allows a wider class of problems to be considered, in particular cases in which the rewards are counts or binary variables using, respectively, Poisson or logistic regression. Like Lin-Ucb, GLM-UCB requires a matrix inversion and can be very costly in terms of time.

Recently, a bandit algorithm based on tree regression has been proposed, called the BanditForest algorithm [24]. This algorithm uniformly assigns items to each arm until a tree forest models the link function. This random forest is built from the joint distribution of contexts and rewards. Thus, all past observations (context and rewards) must be stored. The uniform assignment leads to excessive selection of sub optimal arms, causing the algorithm's performance to suffer. Moreover, the main limitation of the algorithm is that it depends on four parameters, and therefore requires strong domain expertise: two parameters directly influence the level of exploration, one controls the depth of the trees, and one determines the number of trees in the forest [25]. In [26,25,**?**] the authors propose the tree bootstrap algorithm based on a similar approach but parameterless, i.e. tree depth is automatically determined. However, these algorithms only consider binary rewards, which strongly limits their use.

The kernelised stochastic contextual bandit Kernel-Ucb [7] uses reproducing kernel Hilbert space (RKHS) to provide a non-linear model of the link reward function (like GLM-UCB) but can be slow in arriving at a decision.

In addition to making assumptions (e.g., Gaussian distribution, binary reward ...) in order to remain understandable and implementable, the literature identifies the following drawbacks.

– In [27] the authors explain that several dynamic allocation strategies do not provide more benefits than frequentist allocation when the assumptions

(e.g., linear dependence between characteristics and reward, independence between items, ...) are not valid.
- The choices made by the algorithm are not explicit (black box). While understanding choices is not always necessary in a recommendation system, it is important in case of A/B-Tests as the user seeks to understand why and for whom one variation is better than another.
- These strategies often require a large dataset.
- The CPU and/or memory requirements can be significant, particularly when the difference between versions is small.

Two step-based approaches In two step-based approaches, it is assumed that there exist natural groups, each having a Gaussian reward distribution and that these groups can be determined before the A/B-Test itself. The idea is to build these groups *a priori*. When a new item is submitted to the system, it is first automatically classed into a group. Then, it is assigned to an arm considering the group the item belongs to.

In [28], the authors propose the Single-K-UCB strategy and show that if groups are well defined, the cumulative regret converges asymptotically early in the process and the average regret falls significantly. Indeed, intuitively, the cumulative regret is in this case bounded by the sum of the cumulative gaps between the best arm and sub-optimal arms for each group (which is higher than the gap of a non informative strategy). The authors assume that the reward distributions are clustered and the clusters are determined by several latent variables. They assume that there is a surjective function f that links each item (with a context c_t) to a group k, i.e. $f(c_t) = k$, such that the reward distribution of a group k applied to an arm a, $\nu_{a,f(c)}$, is σ-Gaussian (where σ^2 is the variance). Unfortunately, they do not specify how to identify f and how to obtain the groups. They only study the problem in a context-free setting and provide a weak performance guarantee when the reward distribution is unknown in the clusters [29].

To address this problem, we propose a new method called Ctree-Ucb, which is detailed in the next section.

4 Ctree-Ucb: a contextual approach to A/B-Tests

4.1 Ctree-Ucb process

Our contribution aims at addressing the constraints and needs experienced by users in real-world applications. Thus, we first focus on a method that can be applied in real time : for instance, in e-marketing, the delay induced by the test (i.e., the dynamic allocation) must be lower than the usual display time of a webpage. Secondly, we consider that the items submitted to the system can be very heterogeneous but can be clustered according to a given criterion. Finally, we aims for the results of the exploration phase to be understandable by the user, and possibly reusable.

In this context, we propose an approach that consists in defining groups based on item features (i.e., characteristics), each of these groups having as population homogeneous as possible. The main idea is that in such a group, the items have similar behaviors relative to the proposed version and thus a group's reward distribution can be modelled by a Gaussian distribution. Each of these groups can therefore be supported by a non-contextual bandit. The general procedure is that each time a new item is presented to the system, it is automatically assigned to a group according its own features and then, through the associated bandit, a variation is assigned to it. As the complexity of this type of bandit is low, this ensures a satisfactory response time.

In summary, the proposed method Ctree-Ucb consists in two main steps:

- an offline process for creating groups based on the available data collected from the original variation,
- multiple A/B-Tests online.

We illustrate the general idea by an example in which the test concerns the improvement of an existing web page (A). Before starting the test, the behaviours of visitors who have seen page (A) are observed. We assume that the user has collected each visitor's characteristics (browser language, number of visits to the site before arriving on this page, ...) and if a transaction was made after seeing the page. Based on these collected data, our method builds a segmentation model able to classify a visitor into a group using the collected data.

Next, the user constructs variation B (by modification of the original page) and starts the test. In this A/B-Test, a bandit is associated to each group and aims at determining which variation maximises the gains to the visitors classified into its group. Then each visitor arriving on the page is submitted to the test: (1) the visitor is classified into a group by applying the model to its characteristics, (2) the bandit associated to the group dynamically assigns the visitor to a variation, (3) the bandit updates its statistics about the arms.

4.2 Step 1: Offline building of groups and associated classifier

To construct the groups, we suppose that there is information describing the performance of the original version (i.e., the version in production before the test). As such, a database (referred to here in as DB_{init}: $\mathcal{L}$) contains an item's context and reward (conversion, number of clicks, etc.) produced on the original version.

Thus, if the test to be carried out concerns the same type of reward and is on a variation of the original version (referred here as the arm A), the use of this information to build groups can only be beneficial. A model can therefore be learned using training and validation sets extracted from DB_{init}, and used to predict the group of a new item according to its context.

Many supervised methods exist to produce such a model. For instance, decision tree-based algorithms such as C4.5 [30] or C.A.R.T. [31] have shown their effectiveness in finding such homogeneous groups according to a numeric/binary

rewards using an entropy measurement (C4.5) or the Gini index (C.A.R.T.). Unfortunately, they present two fundamental issues: overfitting and selection bias towards continuous features [32]. Conditional inference tree based approaches have shown their robustness in comparison to these previous algorithms [33], and thus have a high level of stability and robustness [34,**?**].

In the first step of our method, performed offline, a conditional inference tree algorithm called CTREE [35,36,37] is applied to a training dataset (herein referred to as $\mathcal{L}_n$) of n items in order to identify homogeneous groups (Algo. 2). It consists in initially creating one group containing all the items. This group is associated with the root node of the tree. Then, a recursive divisive process is applied to this node. An independence hypothesis H_0 between each $j \in \{1, \ldots, d\}$ feature and the reward distribution are evaluated for all the items of the associated group, then:

- If the hypothesis is rejected, the group is split into two subgroups using the feature with the highest correlation with the reward, j^*, according to the value of this feature that maximises the difference between each group's distribution. The algorithm is recursively applied to the two new nodes associated with the two subgroups.
- If the hypothesis H_0 cannot be rejected at the predetermined risk level ϵ, for any feature, the recursion stops.

To verify the correlation hypothesis, statistical tests exist in the literature (Bravais-Pearson test, Spearman test, Chi-squared test,... [35,36]). At the initialisation of the CTREE-UCB scheme, such correlation tests must be defined according to the feature types (continuous, binary, categorical, ...) and reward type (continuous or binary) [38].

The conditional inference tree does not require a pruning process, which avoids overfitting. Moreover, selecting of the value upon which to split is based on the univariate p-values, thus avoiding a variable selection bias towards characteristics with many possible split values. If a statistically significant observation could have risen by "chance", because of the size of the parameter space to be searched, Bonferroni correction could be applied [39]. However, tests integrating categorical features can require a very long computation time when Bonferroni correction is applied [40]. Nevertheless, Bonferroni correction by the Monte-Carlo method [41] can be used to reduce this time. Such a correction includes a random part, which varies the tree structure.

At the end of Step 1, groups are described by a reward average and defined by one or more features. Using this information, a predictive function f is defined which links each new item to a group. This function predicts a group k defined by an expected reward according to A. Thus, this function f can also be considered as a non-linear regression function.

This regression is based on the method described in [37] using the test statistic $\mathbf{T}$ which is derived from [35]. The appendix gives more details on this method. This function is used during step 2 of the A/B-TEST. As step 1 is performed offline, it does not increase the computational time of CTREE-UCB. Figure 3 shows an example of an obtained regression tree.

Algorithm 2 CTREE algorithm

Require: $-\ \epsilon \in]0,1[$
- A dataset of features Y and response X.
- An influence function h depending on the scale of X.
- An appropriate function g_j, which depends on the scale of the feature Y_j

Calculate the the test statistics s_{j0} for the observed data
Permute the observation in the node
Calculate s for all permutations
Calculate the p-values (number of test statistics s, where $|s| > |s_0|$)
Correct p-values for multiple testing
if H_0 not rejected (p-value $> \epsilon$ for all Y_j) **then return**
else
    Select feature Y_j^* with the strongest association (smallest p-value)
    Search for the best split of Y_j^* (maximize test statistic s) and partition data
    Apply CTREE to both of the new partitions

Output: A hierarchical partitioning

4.3 Step 2: Online A/B-Test

The online step corresponds to the A/B-Test. It consists in classifying each item into a group. The dynamic allocation is then performed by the bandit associated to the group.

As the bandits are independent, each of them can stop the exploration phase at any time and switch to the exploitation mode. The test can then end either when all the bandits are in exploitation mode, after a given number of items, or for a predefined duration. Algorithm 3 defines the Ctree-Ucb method.

The computational complexity of using CTREE to predict an average reward depends on the depth of the tree, and the depth of the tree is proportional to the (base 2) logarithm of the number of leaves [36]. The logarithm of a number grows slowly as that number gets larger; therefore even trees with a very large number of leaves will not be very deep. That makes CTREE very fast in terms of use and computation.

4.4 Example on simulated data

Observations made on the data collected via A/B-Tests indicate that some of the functions linking the rewards and the feature can be modelled by a piece-wise continuous function as for example, the link between the price of a product and the quantity purchased. If the site offers one item for every three items purchased, linear modelling between the feature (quantity of item) and the reward no longer holds. If a treatment is effective for young children and older people in the medical field but not for adults, linear modelling also does not work. However, by using a pairwise function can represent such cases. In such a function, the link is linear only over an interval of values taken by the feature. When the link function between a feature and a reward is linear or piece-wise continuous, the above-mentioned traditional bandit strategies have an increasing cumulative regret. To

Algorithm 3 CTREE-UCB algorithm

Require: $\alpha > 0, DB_{init}, \epsilon \in [0, 1]$

1: Generate a conditional inference tree using DB_{init} and f with an accepted error ϵ using CTREE.
2: **loop**
3: $c_t \leftarrow$ a new item with a vector Y_t of features
4: Assign c_t to group k by $f(c_t) = k$
5: **if** $T_{a,k}(t) = 0$ **then**
6: $A_{t,k} = a$
7: **else**
8: $A_{t,k} = \text{argmax}_{a \in \mathcal{A}} \{\hat{\mu}_{a,k,t} + \alpha \sqrt{\frac{2 * log(\sum_{a \in \mathcal{A}} T_{a,k}(t))}{T_{a,k}(t)}}\}$
9: Assign arm $A_{t,y}$ to c_t
10: $X_{c_t,A_t,k} \leftarrow$ the arm $A_{t,k}$ reward
11: Update $\hat{\mu}_{A_t,k}$ and $T_{A_t,k}(t)$

Output: A sequence of arm choices and rewards for each group k

validate our method, we first propose to simulate data from a pairwise function (9600 features x and 9600 rewards for each arm) and compare the results between CTREE-UCB, LIN-UCB, UCB and RANDOM. We report an example of a simulation that tests the performance of CTREE-UCB under real assumptions. For each variation (A or B), 19200 rewards are generated by the following function, related to a feature X:

$$\theta_A = (2, -1, 1.5, 0),$$
$$\theta_B = (1.5, -0.5, 1.25, 0),$$

$$X = \begin{cases} X_A = \theta_A[1], X_B = \theta_B[1], & \text{if } 1 \leq x_1 < 2 \\ X_A = \theta_A[2], X_B = \theta_B[2], & \text{if } 2 \leq x_1 < 3 \\ X_A = \theta_A[3], X_B = \theta_B[3], & \text{if } 3 \leq x_1 < 4 \\ X_A = \theta_A[4], X_B = \theta_B[4], & \text{if } x_1 < 1 \text{ or } x_1 \geq 4. \end{cases}$$

Offline step We use 30% of the data DB_{init} for training (thus the past rewards of A are the only ones observed), and the remaining 70% are used for the test. The regression tree correctly identifies groups in which the link between the feature and reward is identical (each final leaf is a group, represented by an estimated average, see Figure 3).

A/B-Test (Dynamic allocation) During the A/B-TEST itself, a dynamic allocation is made to each group. Figure 5 shows the cumulative regret over time of different algorithms. Figure 4 show the cumulative regret of CTREE-UCB specific to each group.

The following section presents the same comparison on real datasets.

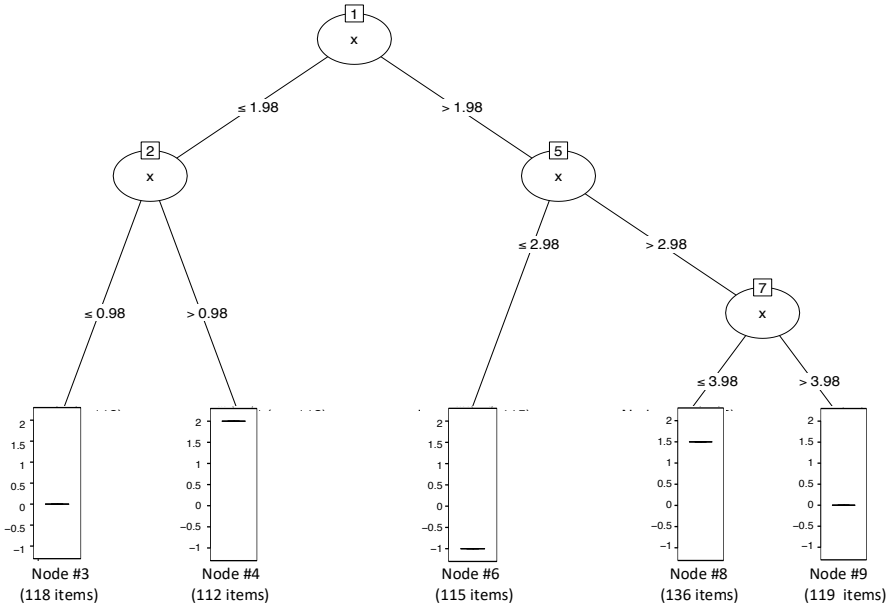
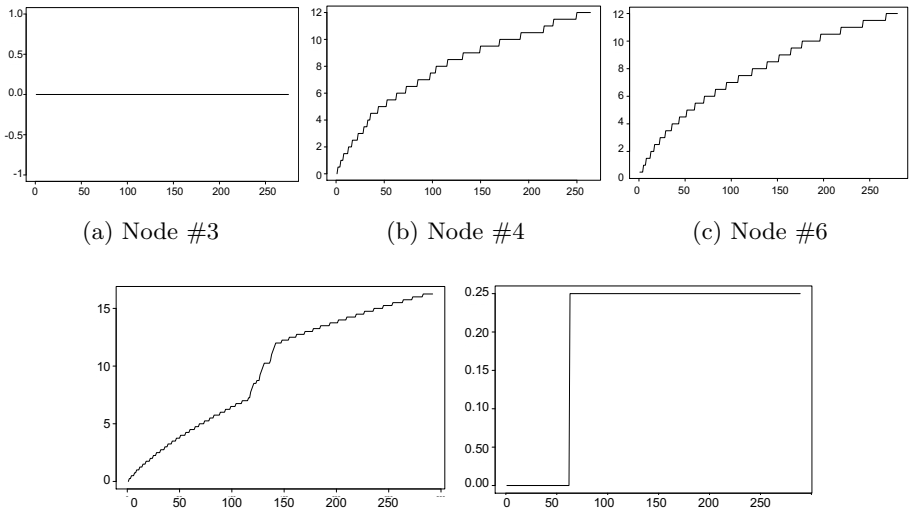


Fig. 3: 5 groups have been identified (600 items)



(a) Node #3 (b) Node #4 (c) Node #6

(d) Node #8 (e) Node #9

Fig. 4: Associated cumulative regret evolution of CTREE-UCB (9000 items)

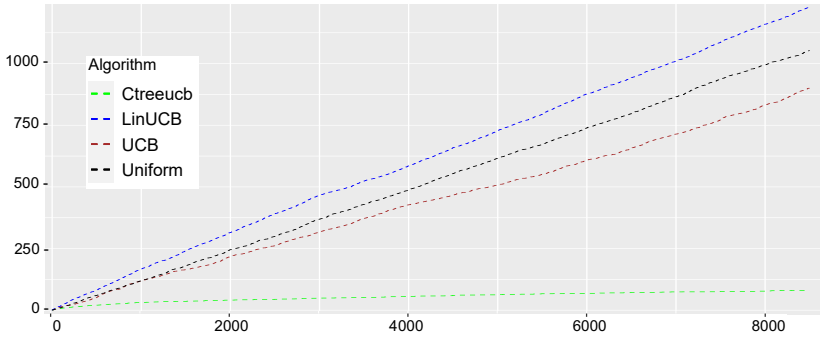


Fig. 5: Cumulative regret evolution with a non linear reward function according to number of submitted items.

5 Materials and experimental setting

To evaluate the performance of CTREE-UCB, we compare it to existing A/B strategies: a global-based bandit (UCB), the LIN-UCB and KERNEL-UCB bandits, as well as a RANDOM-based algorithm that chooses variations alternatively.

The main criteria for this evaluation are the cumulative and average regret. The experiments are carried out over three data sets: the first a public data set, the others are from AB Tasty and correspond to e-merchant A/B-TESTS. All experiments are carried out using the R programming language on a Intel® Core™ i5-8250U CPU with 8 threads running at 1.60 GHz with 7.5 GB of RAM on 64-bit Ubuntu 17.10. All materials (including data, the conditional tree regression framework `CTREE` [38] and CTREE-UCB) are available from: `https://github.com/manuclaeys/bandit4abtest`.

5.1 Data

Small MovieLens dataset This dataset comes from the IMDB public database[7]. It contains movies described by 14 binary characteristics (Adventure, Action, Comedy, Drama, Thriller, Romance, Sci-Fi ...) and their associated ratings (from 0 to 500) given by film reviewers. To simulate an A/B-TEST using this data, we define:

- Movies as items: There are 9125 movies in the original database.
- Film reviewers as the variations denoted by A, B, C, D and E corresponding to 5 reviewers.
- Ratings as the rewards: The reward associated to a movie c_t is the rating $R_r(c_t)$ given by reviewer r associated to the variation. In case the film has not been rated by this reviewer (which may appear with recent films), the average

[7] These datasets change over time but the former version that was used in our experiments is available from: `https://github.com/manuclaeys/bandit4abtest`

of all other reviews evaluates the missing value. Therefore, $X_{\cdot,t} = R_r(t)$ if $R_r(t)$ exists in the dataset.

The objective is to obtain the best cumulative film evaluation.

AB Tasty database The AB Tasty database comes from A/B-Tests consisting in comparisons of a variation of a web-page with its original state (referred here as P_2 and P_1, respectively. These tests were performed in 2018 by AB Tasty itself for several e-merchant clients. They consisted in using a static allocation with an equal distribution of visitors between the two pages P_1 and P_2.

For each test, visitors who navigate on the tested web page (*i.e.*, a potential customer) are identified by an ID: the first time they visit the web page, a new visitor description (see Table 1) and its associated ID are generated. This description may vary depending on the data available to the user. Then a variation (A or B) is allocated to it. A cookie memorizes the description, the ID, and the assigned version. Each time they return to the tested page, the same variation is shown. Therefore we assume that no statistical associations between visitors exist. All visitor actions during their visits are stored. At the end of the test (i.e., after T visitors) each visitor's reward is computed (and collected) as the visitor's purchase value during all their visits regardless of the visit(s) the purchase(s) happened in after the assignment. It is defined by cumulative sum if the visitor c_t has made a purchase on the web page.

For experiments, we have extracted two datasets from this database: ABt_1, which concerns a clothing sales website (8477 items) and ABt_2, which concerns a media website (2265 items). Their objective is to increase the value of a purchase. In the sequel, we present the results of these two A/B-Tests performed on two different websites.

Type	Features (number of possible values or domains)
Integer	Visits ($\mathbb{N}$)
Categorial	Navigator's language (27), Navigator type (6), Device (3), Operating System (7)

Table 1: Item features of the A/B-Test dataset.

Notations used throughout the paper: For N variations $V_0, V_1, \ldots, V_N$, S_i is the set of items to which variation V_i has been allocated, $T_i = |S_i|$ and $T = T_0 + T_1 + \cdots + T_N$ where $|.|$ denotes the cardinality of a set.

5.2 Comparison methods

Ctree-Ucb performance is compared to four algorithms:

- Two algorithms with a non informative strategy, which do not take into account the item's context: RANDOM which is parameter free and the UCB strategy described in Section 3.1.
- Two algorithms with a contextual strategy (Section 3.2). The LIN-UCB algorithm using a linear reliability assumption and the KERNEL-UCB algorithm without a linear reliability assumption: This policy estimates each variation's reward, in addition to a kernel regression of characteristics.

Each algorithm will provide a sequence of choices. These sequences are compared to a model that always chooses the best variation (see Section 2.1), trained with all the data in the test set, so the cumulative and average regret at the end of the A/B-TEST (iteration T) is evaluated.

We note that LIN-UCB and KERNEL-UCB require the transformation of categorical characteristics into binary values.

5.3 Experimental protocol

The A/B-Test parameters All the algorithms (except RANDOM) are derived from the standard UCB algorithm, which requires the setting of the confidence interval parameter α (Section 3.1). To evaluate the impact of this parameter on the results, we carried out experiments with different values of α (from 0.25 to 2.5, as generally found in the literature).

To evaluate the impact of this parameter on the results, we carried out experiments with different values of α (from 0.25 to 2.5, as generally found in the literature).

To limit the CPU time consumed by the KERNEL-UCB algorithm, we limited the number of items used in the kernel regression to 100.

A/B-Test simulation The simulation principle is to apply each algorithm on data sets and compare their obtained cumulative regrets. We compare the results with a non-linear regression (CTREE) model that learns from all the data. Thus, when assigning an item to a variation, the regret is evaluated as the difference between the maximum prediction (from all the possible variations) and the prediction of the chosen variation. This assessment can be seen as the difference between conditional averages and is based on the theoretical definition presented in Section 2.

For the CTREE-UCB method, the offline step (see Section 5.3) is performed first and consists in learning a conditional regression tree. Then each item is evaluated by this tree to determine which group it belongs. Finally, the item is submitted to the classical UCB algorithm associated with this group.

Since a tree is built using data from the original variation, the choice of variation A affects the rest of the process. As such, we considered each variation as a potential original variation for each data set. . Therefore, for the MovieLens dataset, five configurations was tested, each corresponding to a different choice of movie rating as the original variation. In the same way, each page was tested as the original web page (A).

Ctree-Ucb offline step The offline step of Ctree-Ucb consists in defining the item groups used in the contextual A/B-Test. It is crucial as it has a large impact on the A/B-Test process. To produce these groups, we used the R conditional tree regression framework CTREE [38] with a ten-fold cross-validation and different maximum error risk. The displayed tree represents the groups graphically by an expected average and optionally (depending on the user's choice) the distribution boxplot.

To assess this impact, we carried out experiments with different configurations of the ratio between the number of items used to learn the regression tree and those used to simulate the A/B-Test.

Additional notations: $L = |\mathcal{L}|$, where $\mathcal{L}$ is the set used to learn the conditional regression tree. $T_{A/B} = |S_{A/B}|$, where $S_{A/B}$ is the set used to simulate the A/B-Test. ϵ is the error risk parameter to CTREE.

In order to satisfy the assumption that, prior to the A/B-Test, the user only knows the rewards obtained by the original variation (denoted here by V_A), the regression tree can only be constructed from the set of elements S_A to which the variation V_A has been attributed, thus $\mathcal{L} \subset S_A$.

Two configurations were tested:

- $\text{Conf}_{30,70}$: $\mathcal{L} = 30\%$ of V_0, $S_{A/B} = \sum_i \{70\% \text{ of } S_i\}$,
- $\text{Conf}_{100,100}$: $\mathcal{L} = V_0$, $S_{A/B} = \sum_i S_i$.

Experimental configurations Table 2 summarises the parameters and their potential values.

Data set	Algorithms	Parameters
MovieLens dataset	Ucb, Lin-Ucb, Kernel-Ucb	$\alpha \in \{0, 0.25, 0.5, 1, 1.5, 2, 2.5\}$
ATt_1 dataset	Ctree-Ucb	$V_A \in \{V_i\}$ Config. $\in \{\text{Conf}_{30,70}, \text{Conf}_{100,100}\}$ $\epsilon \in \{0.01, 0.05, 0.1\}$ $\alpha \in \{0, 0.25, 0.5, 1, 1.5, 2, 2.5\}$
ATt_2 dataset		

Table 2: Algorithm parameters.

There are 441 combinations: $3 \times 3 \times 7$ combinations for the 3 UCB-based algorithms and $(2 \times 2 \times 2 \times 3 \times 7) + (5 \times 2 \times 3 \times 7)$ for the Ctree-Ucb method. For the sake of clarity, we report only 88 combinations in our experiments (cf. Tab 3). Tab 3a (resp. 3c and 3b) summarizes the cumulative and average regret of all the algorithms for MovieLens (resp. ABt_1 and ABt_2) dataset.

6 Experiments

6.1 MovieLens dataset

Offline step In our experiments, with Config.$_{30,70}$, 2737 items are dedicated to learning (Step #1) and 6388 items are tested (Step #2), while with Config.$_{100,100}$,

both step use all the 9125 items. Figure 6 shows the obtained tree in Config.$_{30,70}$ where each leaf of the tree associates an average to an identified group. The tree leaves represent the groups identified by CTREE that will then be used in the classification model in the dynamic allocation step.

From the rewards given by reviewer #1 before the test, 9 leaves were generated, which corresponds to 9 groups of items with statistically different distributions of rewards. `Node`#15 is the most represented one with 1133 items. The group that maximises rewards for variation A is group `Node`#11. We also note, for example, that reviewer #1 generally gives a higher rating if the film is in the "Film noir" category (`Node`#11). The lowest average reward is given to "Comedy" (`Node`#8) or "Action" (`Node`#7) movies.

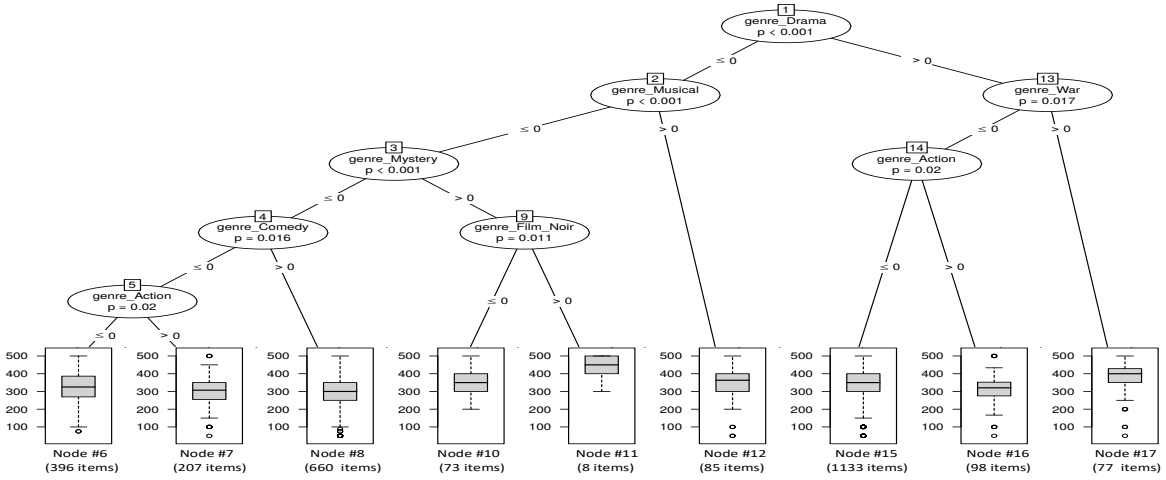


Fig. 6: Conditional inference tree - MovieLens dataset (Config.$_{30,70}, V_A$: Reviewer $1, \epsilon = 0.05$, 2737 items).

A/B-Test (Dynamic allocation) In the MovieLens dataset, some reviewers have not seen many films; therefore, replacing missing scores (films without score) results in identical scores between several reviewers. Therefore, there are films for which the simple regret will be equal to zero regardless of the chosen reviewer. The difference between the averages Δ_a, $\forall a \neq a^*$, being very small, finding the reviewer who maximises rewards (following a context) is difficult (Section 3.1). Such a situation can reduce the performance of the tested bandit algorithms and be comparable to a static allocation strategy (performed by the RANDOM algorithm). The ϵ parameter has a slight influence on the results (regret), and its influence on the results is covered in the later experiments. For

		Ctree-Ucb					Lin-Ucb	Kernel-Ucb	Ucb	random
		$V_A : \mathrm{R}_1$	$V_A : \mathrm{R}_2$	$V_A : \mathrm{R}_3$	$V_A : \mathrm{R}_4$	$V_A : \mathrm{R}_5$				
Configuration		$\epsilon = 0.05$	$\epsilon = 0.05$	$\epsilon = 0.05$	$\epsilon = 0.05$	$\epsilon = 0.05$				
$\text{Config.}_{30,70}$	R_T	**19155**	21628	28458	27894	19976	22378	21239	20650	22808
	$\mathbb{E}[R_T]$	**3**	3.39	4.47	4.37	3.13	3.50	3.32	3.23	3.57
$\text{Config.}_{100,100}$	R_T	**27875**	34152	35679	35908	37679	68101	40848	31146	45643
	$\mathbb{E}[R_T]$	**3.05**	3.74	3.91	3.94	4.13	7.46	4.48	4.41	5

(a) MovieLens dataset (9125 Items)

		Ctree-Ucb						Lin-Ucb	Kernel-Ucb	Ucb	random
		$V_A : P_1$			$V_A : P_2$						
Configuration		$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$				
$\text{Conf}_{30,70}$	R_T	**100**	**100**	**100**	355	355	355	364	592	408	6610
	$\mathbb{E}[R_T] * 10^{-2}$	**1.68**	**1.68**	**1.68**	5.59	5.59	5.59	6.11	9.97	6.87	1.11
$\text{Conf}_{100,100}$	R_T	152	152	152	67	67	67	**24**	448	241	8148
	$\mathbb{E}[R_T] * 10^{-2}$	1.79	1.79	1.79	0.79	0.79	0.79	**0.28**	5.28	2.840.79	0.96

(b) ATt_1 dataset (8477 Items).

		Ctree-Ucb						Lin-Ucb	Kernel-Ucb	Ucb	random
		$V_A : P_1$			$V_A : P_2$						
Configuration		$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$				
$\text{Conf}_{30,70}$	R_T	**1251**	1263	1265	3110	3110	3110	3092	4469	4284	4279
	$\mathbb{E}[R_T]$	**0.67**	0.67	0.67	1.67	1.67	1.67	1.67	2.40	2.30	2.30
$\text{Conf}_{100,100}$	R_T	**1994**	**1994**	**1994**	3843	3843	3843	2457	7007	6114	6862
	$\mathbb{E}[R_T]$	**0.75**	**0.75**	**0.75**	1.44	1.44	1.44	0.92	2.63	2.30	2.57

(c) ATt_2 dataset (2265 Items).

The best performances (cumulative regret , average) appear in bold in each table
Table 3: Influence of segmentation parameters on cumulative regret (R_T) and average regret ($\mathbb{E}[R_T]$) ($\alpha = 1$).

readability of Table 3, only the classic value of the accepted error risk is reported, i.e. $\epsilon = 0.05$.

Ctree-Ucb has the lowest cumulative regret (in bold in Table 3). The Lin-Ucb algorithm has a weak performance, one explanation could be that the linearity assumption required by this algorithm is not valid. Kernel-Ucb has a cumulative regret comparable to a static allocation (random). More data is probably needed to complete its regression. Ctree-Ucb yields better results with reviewer #1 (in bold) with Config.$_{30,70}$ or Config.$_{100,100}$. Learning on reviewer #3 (on Config.$_{30.70}$) or reviewer #5 (Config.$_{100.100}$) decreases its performance. We assume that this implies an over- or under-learning depending on the configuration.

To get good results with Ctree-Ucb, whatever the V_A parameter, the offline step must be performed on a population representative of the one to be tested.

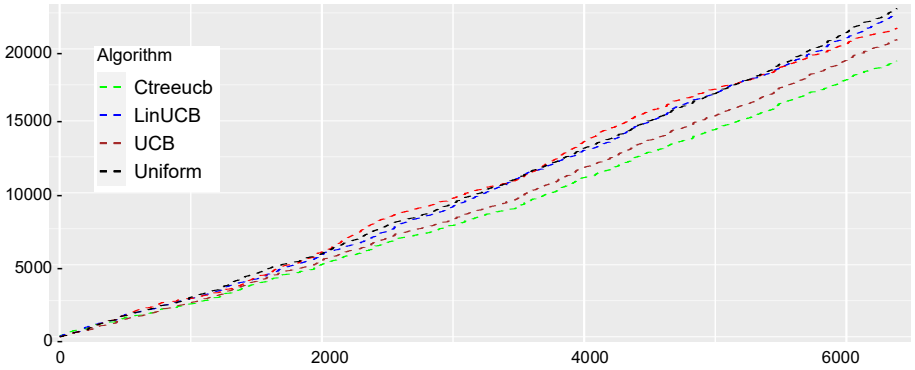
Figure 7a shows the cumulative regret of each algorithm for a given configuration. The lowest regret during the test is that of Ctree-Ucb (in green). The strategy Ucb (in brown) comes in the second position in terms of performance. The highest cumulative regret is that of random (in black). However, these results indicate a linear regret for all algorithms.

Figure 7b shows how α affects cumulative regret. In this experiment, except for Kernel-Ucb, the value of α has a low influence on the cumulative regret of the studied algorithms. Whatever the value α, Ctree-Ucb produces the best results and guarantees their stability. Unlike other algorithms, Kernel-Ucb works differently depending on α. Its cumulative regret seems highly dependent on this parameter. Choosing a sub-optimal value for α can therefore make it less effective than random (Fig. 7b).

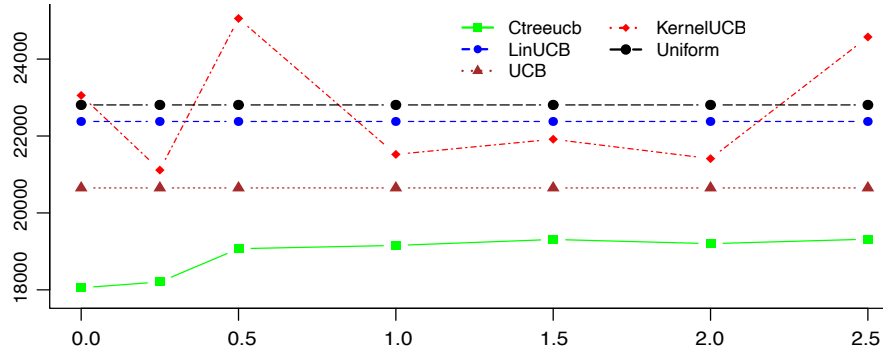
6.2 $\mathbf{ABt_1}$ dataset

Offline step In our experiments, with Config.$_{30,70}$, 2543 visitors are dedicated to learning (Step #1) and 5934 visitors are tested (Step #2) while with Config.$_{100,100}$, both steps use all the 8477 visitors. Fig. 8 shows the obtained tree in Config.$_{30,70}$. In each leaf is reported the expected reward (a purchase value). The first step identifies 10 group. The number of past visits (`visit.y` : visits until yesterday) before seeing the tested page has the strongest correlation with the purchase values. However, purchase value can increase or decrease according to the visitor's user agent or language.

A/B-Test (Dynamic allocation) On Config.$_{30,70}$, all results provided by Ctree-Ucb (in bold) are the best. Table 3b gives the cumulative regret according to the different parameters (ϵ and V_A) for the ATt_1 dataset. With all configurations, the ϵ parameter does not modify the tree structure. However, the challenge in step 1 is to avoid overfitting (too many groups, as in Config.$_{100,100}, V\_A : P\_1$ in Table 3b) or underfitting (too fewer groups, as in Config.$_{30,70}, V\_A : P\_2$ in Table 3b). In fact, too few groups leads to a performance similar to that of a non-contextual strategy. On the other hand, too



(a) According to number of items ($\alpha = 0.25$)



(b) According to α (6388 items)

Fig. 7: Cumulative regret evolution for MovieLens dataset (V_A : R_1, $Conf_{30,70}$, $\epsilon = 0.05$)

.

many groups slows down the exploration period, although CTREE-UCB's results remain more effective than those of UCB, KERNEL-UCB, and RANDOM.

In this test, the variation A was the best for all versions regardless any characteristics. We can conclude that all the tested bandit algorithms have a logarithmic cumulative regret (see Fig. 9a). Finally, because the exploration period can be stopped earlier (after 2000 visitors) with our method when compared to a frequentist approach (RANDOM), the difference on the accumulated regret is here more than 5000 euros.

6.3 ABt_2 dataset

Offline step In our experiments, with $Config._{30,70}$, 679 visitors are dedicated to learning (Step #1) and 1586 visitors are tested (Step #2) while with $Config._{100,100}$, both steps use all the 2265 visitors.With $Config._{30,70}$, 7 groups are discovered in

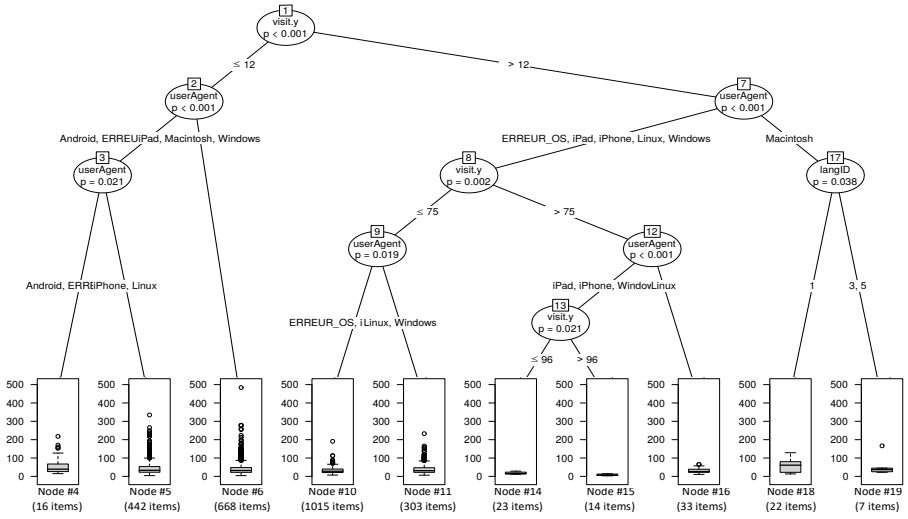


Fig. 8: Tree from ATt_1 dataset ($\text{Config}_{\cdot 30,70}, V\_A : P\_1, \epsilon = 0.05$, 2543 items).

the first step. We note that the characteristics present in the previous experiment (user agent, visits) also influence groups in this experiment. In addition, the browser (called `name`) becomes relevant in this experiment. We hypothesise that since the sale concerns online videos, the user's browser can likely influence the visual rendering.

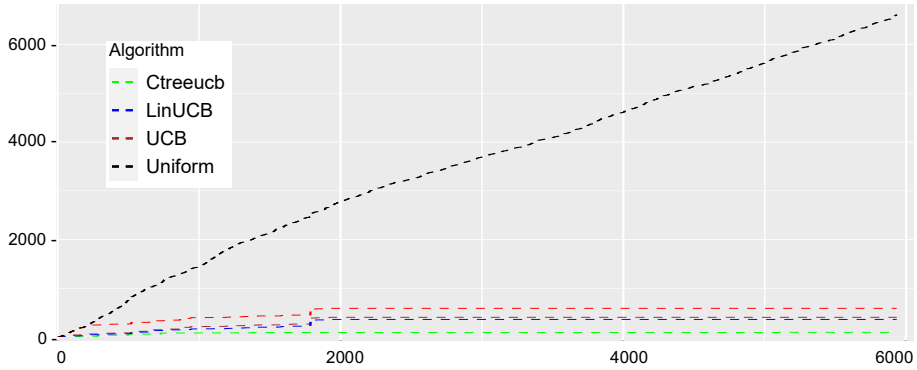
A/B-Test (Dynamic allocation) Table 3c gives the cumulative regret according to the different parameters (ϵ and V_A). On $V_A : P_1$, $\text{Conf}_{30,70}$ the tree structure is only slightly modified (occurrence or avoidance of a maximum of one/two groups). On $\text{Conf}_{100,100}$ with all configurations the ϵ parameter does not modify the tree structure (Figure 10b). However, according to $V_A = P_1$ Ctree-Ucb gives the best results whatever the configuration. Nevertheless, on $V_A = P_2$, the best results are given by Lin-Ucb. Figure 10a presents the cumulative regret according to α and shows the robust performance of Ctree-Ucb.

Group analysis:

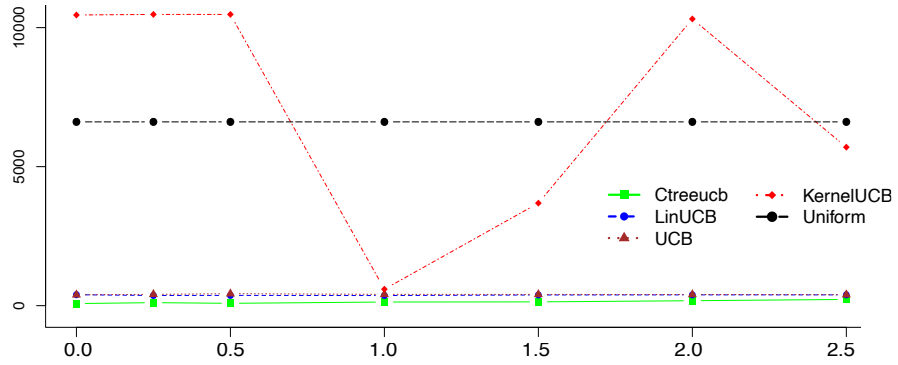
The cumulative regret of Ctree-Ucb (Fig. 10b) is the sum of the cumulative regret of each group. W propose a more detailed analysis of Ctree-Ucb's results by observing the cumulative regret of the 7 subgroups. The parameters of Ctree-Ucb are: $\text{Conf}_{100,100}$ $V_A = P_1$, $\epsilon = 0.05$, $\alpha = 1$.

We note that the group names in Fig. 11 refer to the id of the leaf in the tree and not in the n-th group.

– Figures 11b, 11c, and 11e show that the cumulative regret of Group #5, Group #7, and Group #11 converges asymptotically. For example, in Group



(a) According to number of items ($\alpha = 0.25$)



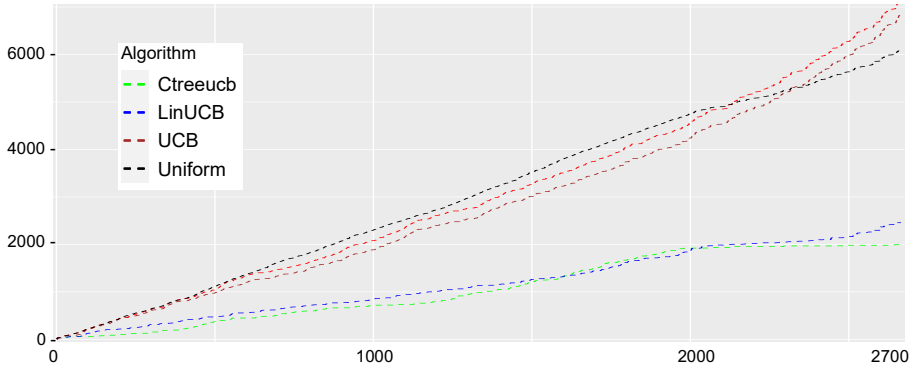
(b) According to α (5934 items)

Fig. 9: Conditional inference tree - ATt_1 dataset ($V_A : P_1$, $\text{Conf}_{30,70}$, $\epsilon = 0.05$).

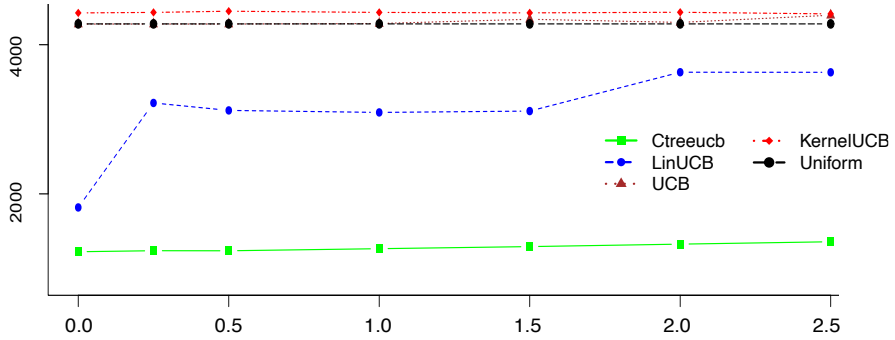
#7, the first half of the visitors tested produced 100% of the total cumulative regret. For the last visitors, their regret is always equal to zero. This result shows that for these groups, CTREE-UCB ends the exploration in an optimal way. This also suggests that these groups are homogeneous (Section 3.1).

– Figures 11a and 11f show each groups cumulative regret, which is almost equal throughout the A/B-TEST. Only a few visitors belonging to this group were impacted by the A/B-TEST. These results show that CTREE-UCB separates unaffected visitors correctly (Section 5.1).
– Figure 11d shows a case in which the cumulative regret grows almost linearly throughout the A/B-TEST. For this group, the variation chosen for exploration either requires more items or is not the best for all visitors. Different reasons may explain this: the gap between the variation's average is very small, learning from the original page did not correctly identify all possible groups existing in the test dataset, or the characteristics used are not sufficient to give a reliable average for this group.

Time analysis:



(a) According to number of items ($\alpha = 0.25$)



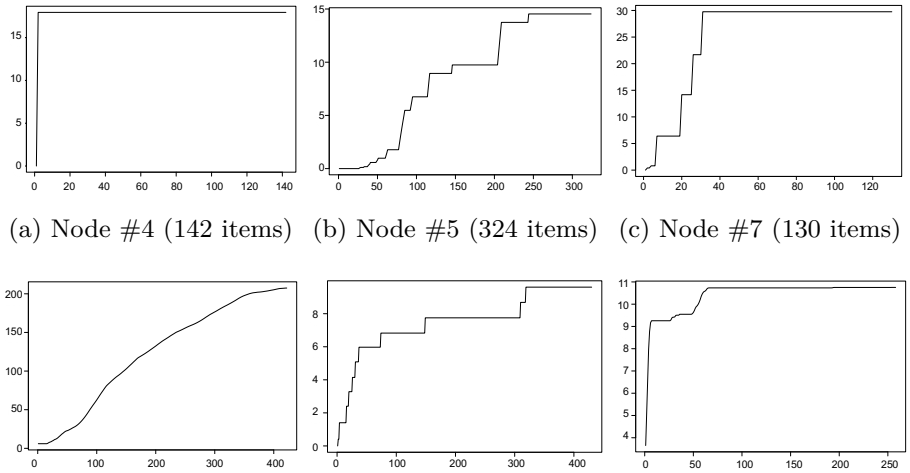
(b) According to α (2265 items)

Fig. 10: Cumulative regret evolution - ATt_2 dataset ($V_A : P_1$, $\text{Conf}_{100,100}$, $\epsilon = 0.05$).

We report in Tab. 4 the computation time required for CTREE-UCB with the three datasets (Config.$_{100,100}$).

Dataset	#features	Size	Step #1	Step #2
MovieLens	19	18250	0.297s	0.347s
ABt_1	5	8477	0.117s	0.523s
ABt_2	4	2265	0.36 s	0.504s

Table 4: Computation times - CTREE-UCB (Config.$_{100,100}$).



(a) Node #4 (142 items) (b) Node #5 (324 items) (c) Node #7 (130 items)

(d) Node #8 (422 items) (e) Node #11 (430 items) (f) Node #13 (258 items)

Fig. 11: Cumulative regret evolution for 6 groups out of 7 identified (representing 1706 of the 2265 items tested) from ABt_2 dataset ($\text{Conf}_{100,100}$ $V_A = P_1$, $\epsilon = 0.05$, $\alpha = 1$).

Step #1 of Ctree-Ucb is performed before the A/B-Test itself (offline). Thus, the time allocated to this step does not impact the test itself and can be ignored.

We note that as each bandit works independently, it seems easy to reduce the computation time by using a multicpu computer. Moreover, the most important for us is the ability of the system to make the assignment of arm in real time. Indeed, for example, the choice between two versions of a web page cannot suffer any delay: for each visitor, the algorithm must choose the variation in less than half a millisecond to avoid delays in displaying the page.

Tab 5 details and compares time responses of the four algorithms on the ABt_2 dataset. One can see that the response time per visitor for Ctree-Ucb is less than one millisecond and outperforms all the other algorithms, except Ucb. However, as we introduced in Section 3.1, Ucb is not contextual and tends to produce worse results than Ctree-Ucb. Kernel-Ucb requires the longest computation time. It is mainly due to the regression calculation of the kernel.

Experiments on one million simulated data and on the other datasets show that the behavior of the Ctree-Ucb algorithm is very similar[8].

[8] To limit the paper length, these experiments have not been reported in the paper but can be easily reproduced using data and codes available at: `https://github.com/manuclaeys/bandit4abtest`

	Ctree-Ucb	Lin-Ucb	Kernel-Ucb	Ucb
Total time	0.504s	0.622s	16.013s	0.096s
Max group	0.504s			
Time by item	0.18ms	0.23ms	6ms	0.03ms

Table 5: Calculation times - ABt_2 dataset ($Config._{100,100}$).

7 Discussion

Based on our results, it can be concluded that Ctree-Ucb responds to different A/B-Test issues. Our results show the performance of Ctree-Ucb on different types of tests (continuous or categorical characteristics, continuous reward, and several possible variations higher than two (A/B/C/...).

For reasons of confidentiality, only datasets provided by websites that agreed to publish their data have been presented. If the user wants to improve performance, each group can be processed independently on a server to accelerate computation time (nevertheless it was not necessary given the good results in calculation time for AB Tasty datasets). However, Ctree-Ucb obtained good results on websites that could receive more than 2000 visitors per second.

Ctree-Ucb has three parameters: Conf., V_A, ϵ, and α. From our experiments, we can conclude the following.

- A partial dataset ($Conf_{30,70}$) for step one is sufficient to obtain results comparable to the total dataset ($Conf_{100,100}$).
- By considering different original variations V_A, the results of Ctree-Ucb may be different. However, Ctree-Ucb results remain good compared to Lin-Ucb, Kernel-Ucb, and Ucb.
- The parameter ϵ (associated with the accepted risk in the inference tree) has a low influence on the results, therefore the default value of 0.05 can be used.
- An incorrect α value can lead to a degradation of Ucb performance while Ctree-Ucb is less sensitive to the alpha parameter.

Ctree-Ucb has the following advantages.

- The model can handle both numerical and categorical values. Other techniques are often usable only with specific variable types.
- The construction of groups by a conditional inference tree simplifies their interpretation. Using Boolean logic, the user understands which characteristics have the highest impact on the distribution of a group's rewards, unlike black box models such as neural networks, whose results are difficult to explain.
- Group construction, performed offline, results in a response time comparable to a non-contextual strategy and can be decreased with distributed computing (e.g., one group per server). Thus, when the user wants to have a rapid choice, Ctree-Ucb can be used.

However, Ctree-Ucb requires:

- an original variation, set up before the test;
- stationary reward distributions, as Lin-Ucb, Kernel-Ucb, and Ucb;
- the population of items used to provide the groups, before starting the test, to be representative of the population of items tested.

The quality of the results obtained from Ucb and Lin-Ucb is significantly different according to the data type. These algorithms can be equivalent to random when their assumptions (linearity, ...) are not verified, or when α's value is not optimal.

Kernel-Ucb is challenging to use in practice. As Cesa-Bianch *et al.* [42] note :"when the number of kernel evaluations is bounded, there are cases where no algorithm attains performance better than a trivial sub-sampling strategy, where most of the data is thrown away. Also, no algorithm can work well when the regularisation parameter is sufficiently small or the norm constraint is sufficiently large". Ctree-Ucb can avoid these parameters that can be wrongly chosen. Moreover, with AB Tasty datasets, Ctree-Ucb yields the best results. These datasets correspond to our main objective, and the other is presented to provide results on public datasets.

8 Conclusion

In this paper, we presented a new approach of A/B-Test (called Ctree-Ucb) based on bandit models. It proposes to extend the existing test methods to take into in account context of the items to better learn the best arm and so, to quickest leave the exploration phase. The Ctree-Ucb method consists in extracting groups from past items (i.e., before the modification of the entity and so, before the A/B-Test itself). These groups are then used as contexts to the new items: each of them is associated to a group at its arrival. In each group a non-contextual bandit is dedicated to find the optimal arm.

To validate our proposition we carried out experiments on, on one hand, synthetic data and classical benchmarks and, on the other hand, data coming from real e-merchant's website. These experiments show that Ctree-Ucb achieves a cumulative regret comparable to the best performance of current state-of-the-art methods. We also showed that Ctree-Ucb provides good results regardless of the parameters chosen and that a default setting is enough to obtain reliable results. Furthermore, as the main time consuming phase (building of items groups) is processed before the test itself, the computation time required by Ctree-Ucb to make the dynamic allocations is very low. That allows its integration in an industrial environment where rapid response is crucial.

Moreover, experiments have shown that conditional inference tree method is very powerful for extracting pertinent groups, leading to a decrease in cumulative/average regret during the test. An interesting result is that similar groups

are identified by CTREE-UCB even though the learning is done on different variations. Our results suggest a correlation between the reward distribution of the variations. In practice, in most cases, the influence of a change made by a variation is limited. We can therefore create groups on variation A and assume that they are similar on variation B. Finally, group identification can help guide the user in the composition of the test itself. If the modification was irrelevant for a group, another more specific modification can be proposed.

Nevertheless, our method is based on a strong assumption of stationarity of these groups in the short term, i.e. that the distributions and groups do not vary too much between the capture of rewards and items and the construction of the tree, and thus its use in the test. Thus, launching a campaign just before the test would be a perfect counterexample: the reward distribution will be certainly impacted. Similarly, offering more feminine articles on a site visited mainly by men could bring in a new clientele not present at the time the groups were built. This is a strong scientific issue that we will address in our future work.

Finally, although the results are very good, we believe that our method can still be improved. We propose to extend it to be able to handle temporal data such as visitor's navigation timeline before arriving on a test page.

Supports. This research was supported by AB Tasty and ICube laboratory (University of Strasbourg/CNRS).
Acknowledgments We really thank Jérémie Mary for his assistance with bandit concepts and techniques.

References

1. W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3-4, pp. 285–294, 1933.
2. H. Robbins, "Some aspects of the sequential design of experiments," *Bull. Amer. Math. Soc.*, vol. 58, no. 5, pp. 527–535, 09 1952.
3. A. J. C. Gittins and J. C. Gittins, "Bandit processes and dynamic allocation indices," *Journal of the Royal Statistical Society, Series B*, pp. 148–177, 1979.
4. O. Nicol, J. Mary, and P. Preux, "Icml exploration and exploitation challenge: Keep it simple !" in *Journal of Machine Learning Research (JMLR)*, 2012, iJournal.
5. E. Kaufmann, O. Cappé, and A. Garivier, "On the Complexity of A/B Testing," *ArXiv e-prints*.
6. L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. ACM, 2010, pp. 661–670.
7. M. Valko, N. Korda, R. Munos, I. Flaounas, and N. Cristianini, "Finite-time analysis of kernelised contextual bandits," in *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*.
8. W. Chu, L. Li, L. Reyzin, and R. Schapire, "Contextual bandits with linear payoff functions," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 208–214.

9. T. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in Applied Mathematics*, vol. 6, no. 1, pp. 4 – 22, 1985.
10. L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.
11. M. N. Katehakis and A. F. Veinott, "The multi-armed bandit problem: Decomposition and computation," *Mathematics of Operations Research*, no. 2, pp. 262–268.
12. P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, May 2002.
13. R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Networks*, vol. 16, pp. 285–286, 1998.
14. H. Bastani, M. Bayati, and K. Khosravi, "Mostly exploration-free algorithms for contextual bandits," 2017.
15. M. Tokic and G. Palm.
16. M. N. Katehakis and H. Robbins, "Sequential choice from several populations," vol. 92, no. 19, pp. 8584–8585, 1995.
17. G. Burtini, J. Loeppky, and R. Lawrence, "A survey of online experiment design with the stochastic multi-armed bandit," *CoRR*, vol. abs/1510.00757, 2015.
18. N. Carrara, E. Leurent, R. Laroche, T. Urvoy, O.-A. Maillard, and O. Pietquin, "Budgeted Reinforcement Learning in Continuous State Space."
19. A. Pacchiano, M. Phan, Y. Abbasi-Yadkori, A. Rao, J. Zimmert, T. Lattimore, and C. Szepesvari, "Model selection in contextual stochastic bandit problems," 2020.
20. L. Zhou, "A survey on contextual multi-armed bandits," *CoRR*, vol. abs/1508.03326, 2015.
21. T. Lattimore and C. Szepesvári, *Bandit Algorithms*. Cambridge University Press, 2020.
22. S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends® in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.
23. S. Filippi, O. Cappe, A. Garivier, and C. Szepesvári, "Parametric bandits: The generalized linear case," in *Advances in Neural Information Processing Systems 23*.
24. R. Féraud, R. Allesiardo, T. Urvoy, and F. Clérot, "Random forest for the contextual bandit problem," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, 2016, pp. 93–101.
25. A. N. Elmachtoub, R. McNellis, S. Oh, and M. Petrik, "A practical method for solving contextual bandit problems using decision trees," in *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*, 2017.
26. ——, "A practical method for solving contextual bandit problems using decision trees," *CoRR*, vol. abs/1706.04687, 2017.
27. J. Vermorel and M. Mohri.
28. O.-A. Maillard and S. Mannor, "Latent Bandits." Jan. 2014, extended version of the paper accepted to ICML 2014 (paper and supplementary material).
29. Y. Qi, Q. Wu, H. Wang, J. Tang, and M. Sun, "Bandit learning with implicit feedback," in *Advances in Neural Information Processing Systems 31*.
30. J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.
31. L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*, ser. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

32. Y.-S. Shih, "A note on split selection bias in classification trees," *Computational Statistics & Data Analysis*, vol. 45, no. 3, pp. 457–466, 2004.
33. J. Mingers, "Expert systems-rule induction with statistical data," *The Journal of the Operational Research Society*, vol. 38, no. 1, pp. 39–47, 1987.
34. C. Strobl, J. C. Malley, and G. Tutz, "An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests." *Psychological methods*, vol. 14 4, pp. 323–48, 2009.
35. H. Strasser and C. Weber, "On the asymptotic theory of permutation statistics," 1999.
36. T. Hothorn, K. Hornik, M. A. van de Wiel, and A. Zeileis, "A lego system for conditional inference," *The American Statistician*, vol. 60, no. 3, pp. 257–263, 2006.
37. T. Hothorn, K. Hornik, and A. Zeileis, "Unbiased recursive partitioning: A conditional inference framework," *Journal of Computational and Graphical Statistics*, vol. 15, no. 3, pp. 651–674, 2006.
38. T. Hothorn, U. München, K. Hornik, W. Wien, A. Zeileis, and W. Wien, "party: A laboratory for recursive partytioning."
39. O. J. Dunn, "Multiple comparisons among means," *Journal of the American Statistical Association*, vol. 56, no. 293, pp. 52–64, 1961.
40. T. Hothorn, K. Hornik, M. van de Wiel, and A. Zeileis, "Implementing a class of permutation tests: The coin package," *Journal of Statistical Software, Articles*, vol. 28, no. 8, pp. 1–23, 2008.
41. T. Morikawa, A. Terao, and M. Iwasaki, "Power evaluation of various modified bonferroni procedures by a monte carlo study," *Journal of Biopharmaceutical Statistics*, vol. 6, no. 3, pp. 343–359, 1996, pMID: 8854237.
42. N. Cesa-Bianchi, Y. Mansour, and O. Shamir, "On the complexity of learning with kernels," *CoRR*, vol. abs/1411.1158, 2014.