Optimization experiments in the continuous space: The limited growth optimistic optimization algorithm

Citation for the original published paper (version of record):
Issa Mattos, D., Mårtensson, E., Bosch, J. et al (2018). Optimization experiments in the continuous space: The limited growth optimistic optimization
algorithm. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11036 LNCS: 293-308.
http://dx.doi.org/10.1007/978-3-319-99241-9_16

Optimization Experiments in the Continuous Space

The Limited Growth Optimistic Optimization Algorithm

David Issa Mattos[1(✉)], Erling Mårtensson[2], Jan Bosch[1], and Helena Holmström Olsson[3]

[1] Department of Computer Science and Engineering, Chalmers University of Technology, Hörselgången 11, 412 96 Göteborg, Sweden
{davidis, jan.bosch}@chalmers.se

[2] Sony Mobile Communications, Nya Vattentornet, 221 88 Lund, Sweden
erling.martensson@sony.com

[3] Department of Computer Science and Media Technology, Malmö University, Nordenskiöldsgatan, 211 19 Malmö, Sweden
helena.holmstrom.olsson@mah.se

Abstract. Online controlled experiments are extensively used by web-facing companies to validate and optimize their systems, providing a competitive advantage in their business. As the number of experiments scale, companies aim to invest their experimentation resources in larger feature changes and leave the automated techniques to optimize smaller features. Optimization experiments in the continuous space are encompassed in the many-armed bandits class of problems. Although previous research provides algorithms for solving this class of problems, these algorithms were not implemented in real-world online experimentation problems and do not consider the application constraints, such as time to compute a solution, selection of a best arm and the estimation of the mean-reward function. This work discusses the online experiments in context of the many-armed bandits class of problems and provides three main contributions: (1) an algorithm modification to include online experiments constraints, (2) implementation of this algorithm in an industrial setting in collaboration with Sony Mobile, and (3) statistical evidence that supports the modification of the algorithm for online experiments scenarios. These contributions support the relevance of the LG-HOO algorithm in the context of optimization experiments and show how the algorithm can be used to support continuous optimization of online systems in stochastic scenarios.

Keywords: Online experiments · Multi-armed bandits
Infinitely many-armed bandits · Continuous-space optimization

1 Introduction

Traditional requirements engineering relies on domain experts and market research to model the user behavior and define requirements for their systems. However, research shows that, often as 70–90% of the time, companies can be wrong about their customer

T. E. Colanzi and P. McMinn (Eds.): SSBSE 2018, LNCS 11036, pp. 293–308, 2018.
https://doi.org/10.1007/978-3-319-99241-9_16

preferences [1–3]. In this scenario, several companies are adding on top their requirements engineering practices the usage of post-deployment data to evaluate the user behavior and set prioritization and optimization objectives. One way use post-deployment data in software development is through online controlled experiments with user behavior. Aligned with a set of long-term business goals metrics [4], these companies are running business-driven experiments, such as A/B tests, to validate their business hypotheses [5].

This movement started with web-facing companies such as Microsoft, Google, Facebook, Amazon, LinkedIn, among others [2, 6–9], and they continuously report the competitive advantages that experimentation techniques such as A/B delivers in business-driven experiments [10]. As these companies scale their experimentation infrastructure and organization to keep their competitive edge, they developed sophisticated techniques to run experiments in range of situations that simple A/B experiments face limitations. Some of these techniques are overlapping experiments [6], optimal ramp-up [9], networked A/B testing [11], multi-armed bandits [12], counterfactual analysis [13] and optimization experiments [14, 15]. With the increasing number of experiments being run every year [1, 9], companies are looking for new techniques that can free some of their research and development resources from the lower risk optimization experiments and allow these resources to be employed in experiments that have higher risk and higher potential return on investment and that cannot be managed automatically by a computer.

In this context, bandit algorithms started to be employed by software companies to simplify the experimentation process in some experiments [12]. Bandit problems is a class of problems that deals with the exploration/exploitation dilemma [16]. This work focuses on a subset of bandit problems called the infinitely many-armed bandit problems. This subset investigates the optimization of parameters in a continuous space, in the presence of an unknown mean reward function. This class of problems is particularly important in online controlled experiments, as several user-behavior assumptions are captured in the systems in the form of constants that can be mapped in a continuous space. The most prominent and least restrictive algorithm for the infinitely many-armed bandit problem is the Hierarchical Optimistic Optimization (HOO) algorithm [17, 18]. However, there is no evidence or empirical evaluation of the usage of this algorithm in online experiments. During the implementation of the HOO algorithm in collaboration with Sony Mobile, this algorithm presented limitations some limitations, such as the computation time, the correctness of the output based on different mean-reward distribution functions, the lack of a criterion to select the best arm at any point, and an estimation of the mean-reward.

The contribution of this work is three-fold. First, we provide a modification of the HOO algorithm to overcome the identified online experiments restrictions, improving the correctness of the output, the time to compute, a criterion to select the best arm and an estimation of the mean-reward function. We call this new algorithm as the Limited Growth Hierarchical Optimistic Optimization algorithm (LG-HOO). Second, we present an implementation of LG-HOO in an industrial setting, in collaboration with Sony Mobile. Third, we provide statistical evidence that supports the modification of the algorithm not only in real-world scenario, but also on simulation scenarios.

This paper is organized as follows. Section 2 presents background information in controlled experiments, bandits problems and the infinitely many-armed bandit problem, the HOO algorithm and related work. Section 3 discusses the research process. Section 4 presents the LG-HOO algorithm, results from simulations, and the results of the implementation of the LG-HOO in an industrial context in collaboration with Sony Mobile. Section 5 discuss the results of the LG-HOO and makes a statistical comparison of the LG-HOO algorithm with the HOO. Section 6 concludes and discusses future research directions.

2 Background and Related Work

2.1 Online Controlled Experiments

Controlled experiments are a technique where the users are randomly assigned to two variants of a product: the control (current system) and the treatment (the system with a change X). The change X can be the implementation of a new feature or the parametrization for optimization of existing features. After the change is implemented, the system is instrumented and the user's behavior and the system performance are computed. After a predetermined period of data collection, the computed metrics for all variations of the system (control and treatments) are analyzed. If the two following conditions are true: (1) the assumption that the external factors are spread out evenly between the two variants due to consistent randomization process holds true (quality checks can help assess this assumption) and (2) the only consistent difference between the treatment and the control is the change X; we can establish a causal relationship of the change X and the observed difference in the metrics. Kohavi et al. [19] provide a detailed guide on how to run online controlled experiments in the web. If the experimentation process is used in incremental variations of the system, this becomes an optimization procedure.

2.2 The Multi-armed Bandit, the χ-Bandit Problem, and the HOO

The Multi-armed Bandit Problem

Multi-armed bandit problems are a class of problems that deals with the exploration and exploitation trade-off. The problem statement and its name come from parallel the of the gambler problem facing a row of N slot machines, also called one-armed bandits. The gambler wants to maximize the end reward from the slot machines after a set of plays. Each slot machine/arm has a fixed unknown probability distribution for the reward. The gambler faces the problem of exploiting the arm that provides the largest reward while exploring other arms to make sure he does not miss arms that can provide a better reward. In its simpler formulation, the bandit has limited number of rows to play, the number of arms is finite, the arms are independent of each other, and each arm has a stationary stochastic distribution over time.

The general problem can be formulated as [13]:

$$a = \pi(\delta),\ \text{Arm } a \in \{a_1 \ldots a_K\} \tag{1}$$

$$y = r(a,\ \delta'),\ \text{Reward } y \in \mathbb{R} \tag{2}$$

Where a is the arm selected, K is the number of arms, π is the user-defined policy (a selection of actions) function to balance the exploration of arms and the exploitation of the best arm so far, δ and δ' are noise variable (that makes the problem stochastic), y is the measured reward and r is the unknown mean-reward function for the selected arm. Several policies can be formulated in this class of problem, the most used is to minimize the regret. Regret is comparison of the cumulative mean reward of the algorithm and the expected reward of playing the optimal arm.

$$Regret(t) = r(a^*) \cdot t - \sum\nolimits_{s=1}^{t} \mu(a_s) \tag{3}$$

Where $\mu(a)$ is the mean reward of the arm a over the time and a^* is the arm with the largest mean-reward over time:

$$a^* = \max_{a \in \{a_1 \ldots a_K\}} \mu(a) \tag{4}$$

The multi-armed bandit algorithms are the construction of the user-defined policy π to select the arm.

The χ-Bandit Problem and The Infinitely Many-Armed Bandit Problem

The χ-bandit problem, also known as the continuum-armed bandit problem, is formulated similarly to the multi-armed bandit problem. However, instead of a predefined and finite number of arms (Arm $a \in \{a_1 \ldots a_K\}$), the χ-bandit problem has an infinite number of arms that are drawn from a continuous set (Arm $a \in \chi$, where $\chi \subset \mathbb{R}$). The χ-bandit problem is part of the general problem of the infinitely many-armed bandits (when the number of arms in much greater than the allowed number of plays). Some of the advantages of selecting the arms from a continuous space compared to the discrete many-arms counterpart are: (1) discretization of the space reduces limits the optimization precision to the discretization interval. To obtain a more refined interval it is necessary to add new arms that have lower confidence compared to the existing ones. (2) It is not necessary to discretize and compute the arms prior to the experiment, as well as keeping statistics for them all. (3) infinitely many-armed bandits require less exploration time than finite armed bandits (discretized) in the same conditions [20].

The χ-bandit problem can be represented as finding the arm a^* that minimizes the regret function:

$$a = \pi(\delta), \text{Arm } a \in \chi, \text{where } \chi \subset \mathbb{R} \tag{5}$$

$$y = r(a,\ \delta'), \text{Reward } y \in \mathbb{R} \tag{6}$$

$$Regret(t) = r(a^*) \cdot t - \sum_{s=1}^{t} \mu(a_s) \tag{7}$$

$$a^* = \arg\min_{a} Regret(t) \tag{8}$$

The infinitely many-armed bandit problems have been studied in different frameworks, Bayesian, frequentist parametric and frequentist non-parametric settings. The Bayesian problem is to compute the optimal actions efficiently, while the frequentist is to achieve a low rate of regret [21]. A class of algorithms for the frequentist non-parametric setting is the hierarchical optimization. A recent report compares Bayesian and the frequentist non-parametric frameworks concluding that major advantage of a hierarchical optimization algorithms is that they are faster in term of time complexity [18]. In the frequentist non-parametric framework two algorithms stand out, the Bandit Algorithm for Smooth Trees (BAST) and the Hierarchical Optimistic Optimization (HOO) [17]. In this work we use the HOO algorithm, as the BAST algorithm makes strong assumptions on the unknown mean-reward distribution functions that might not be valid in real-world applications [17]. An in-depth discussion and comparison with other algorithms are presented in [18, 21].

The HOO algorithm

The Hierarchical Optimistic Optimization (HOO) algorithm [21] investigates the infinite many-armed bandit problem. This algorithm is classified inside of the hierarchical optimization algorithms in the frequentist non-parametric framework. In this section, we briefly describe the HOO algorithm and its assumptions.

The algorithm makes the stochastic assumption of the mean-reward of any new selected arm. This assumption means that the reward from the new arm is an independent sample from a fixed distribution. The reward is assumed to be in the interval of $y \in [0,\ 1]$. This assumption is realistic as the reward metric is defined and can be normalized to this range. The other assumption is that the unknown reward function is continuous around the maximum, which is a reasonable assumption in practical problems [21].

The algorithm aims to estimate the underlying unknown reward function f around its maxima while it estimates loosely f in other parts of the space χ. This is implemented using a binary tree in which each arm is associated to a region of the space. The deeper the tree grows the smaller the subset of the space χ that it estimates. The HOO uses an optimistic estimate B, using the upper confidence bound, for each node. The tree is traversed and at each iteration the node of largest bound B, is selected. Based on the reward the tree is updated.

The algorithm starts from the root and selects the child with the largest bound B (ties are broken randomly) until it reaches the leaf. From the traversed path from root to leaf, it randomly selects one node to play. The node statistics are updated and the tree is extended if it is a leaf. The statistics and bounds are computed recursively from the leaf until the root using the formulas below. Below is the notation used in throughout this work (and is the same as the one presented in [21]).

$v((H,\ I))$ is the value of the node (H, I).

a_{played} is the value of the played arm.

$B_{h,\,i}$ is the bound for the node i at the height h. The children for this node are $B_{h+1,\,2i-1}$ and $B_{h+1,\,2i}$. The root is denoted by the index (0, 1).

n is the current discrete time instance and the mean reward for time is represented by $\widehat{\mu_{h,i}}(n)$.

$T_{h,i}(n)$ is the number of times a node was played until time n.

The bounds are updated according to the formulas below:

$$U_{h,i}(n) = \begin{cases} \widehat{\mu_{h,i}}(n) + \sqrt{\frac{2\ln n}{T_{h,i}(n)}} + v_1\rho^h, \text{ if } T_{h,i}(n) > 0 \\ +\infty, \text{ if } T_{h,i}(n) = 0 \end{cases} \tag{9}$$

$$B_{h,i}(n) = \begin{cases} \min\left\{U_{h,i}(n),\ \max\left\{B_{h+1,2i-1}(n),\ B_{h+1,2i}(n)\right\}\right\} \text{ if } (h,i) \in \text{Tree}_\text{n} \\ +\infty, \text{ otherwise} \end{cases} \tag{10}$$

Apart from the mentioned advantages of χ-armed bandits algorithms in comparison with grid searching, one of the main advantages of this method is the updating of confidence bound of the whole path as a child is selected. Even though a particular node was not played, its confidence bound is updated if any of its descendants are played. This leads to tighter confidence intervals of a whole path. In most grid search and regular multi-armed bandits, the confidence intervals are created and updated only for the discrete played arm.

2.3 Related Work

Optimization in online experiments can be done by using a range of different techniques. The simplest one is conducting sequential A/B/n experiments. This technique has the advantage of having comparable sample sizes for all variations in the statistical analysis at the expense of increase in the regret and the higher sample size for the optimization. Genetic algorithm has also been used in simulation of online experiments. Tamburrelli and Margara [15] proposed an infrastructure and a genetic algorithm to optimize HTML web pages in a large space. However, the proposed solution requires using non-validated assumptions on the hyper-parameters and on the mating strategies. Additionally, the solution requires a large space of unique users that makes it application in real world restricted to very large scale software companies.

Multi-armed bandits algorithms provide a framework for optimization of experiments and it is widely used in industry [14, 22, 23]. Google's Vizier [14] is a tool for black-box optimization that take advantage of multi-armed bandit algorithms. While the paper does not focus on online controlled experiments, it mentions the use of the tool for optimization of web properties such as thumbnail sizes and color scheme. Shang [18] presents an overview of black-box optimization methods using bandits algorithms and Gaussian processes. Mattos et al. [24, 25] presents an architecture framework and architecture decisions to run optimization experiments with a domain specific heuristic for the bandit problem.

3 Research Process

This research was conducted in collaboration with Sony Mobile Communications in Lund, Sweden. Sony Mobile is a subsidiary of Sony Corporation and is a leading global innovator in information technology products for both consumer and professional markets. One of the Sony Mobile's products is transitioning to data-driven development and aims to run experiments continuously throughout its development process. The product consists of a business to business solution, where the user of the software consists of employees of the company that requested the solution. The software development of this product span development for web, mobile, backend systems and distributed embedded hardware. Therefore, the requirements for an experimentation system include the ability of allowing experiments to be run in the variety of supported systems and the capability of supporting both traditional A/B experiments as well as search solutions in a larger or continuous space. An experimentation system (called ACE) that fulfills the requirements was developed following the framework and architecture decisions presented in [24, 25]. A full description of this system is beyond the scope of this paper. During the development of the product several assumptions were made, such as numerical, textual, and GUI constants that has a direct impact in the how the user interact with the system. Some of the numerical assumptions are constants in the real space or in a predetermined range ($x \in \mathbb{R}$ or $x \in [0, 1]$). The development team of this product wants to optimize these constants and to verify these assumptions in based on actual user behavior metrics. The HOO algorithm was selected as the starting point for the optimization search process. The HOO algorithm was implemented in the ACE system and was repeatedly tested and iterated in both simulation and with real users. The results of these iterations were constantly discussed with the product development team and the modifications of this algorithm resulted in the LG-HOO algorithm. The limitations of the HOO algorithm, the changes motivation for the LG-HOO algorithm, and an empirical comparison between both are presented in the Sect. 4.

4 The LG-HOO Algorithm and the Empirical Data

This section presents the modification version of the HOO algorithm [21]. The HOO algorithm was modified to allow its application in online controlled experiments. The LG-HOO follows the same structure as the HOO with the main modifications highlighted below. The growth restrictions motivate the name Limited Growth HOO. The implementation source code, the results for comparison, and the raw data used and the analysis source is available at https://github.com/davidissamattos/LG-HOO.

- A node (arm) is only allowed to grow if it has been played a minimum number of times. This ensures that each arm has a minimum confidence level to ensure the growing in more confident direction. The HOO grows based only on the upper bound of the arm, and this bound can be unrealistic if only one observation has been made, as it grows with $\sqrt{(2 \ln n)/T_{h,i}(n)}$. The tradeoff of selecting a minimum growth limit is that it needs a higher number of plays to reach the same level of

interval refinements (that is related to the height of the tree). However, as shown later, the minimum growth does not imply that the estimated best arm is further to the theoretical best arm when the underlying function is known.

- The original HOO does not point a method for selecting the best arm, as it is intended to be a continuous process. The algorithm indicates that the highest node on the tree represents the maximum of the underlying function. However, in online experiments, after a period of time the company might want to stop the experiment to save resources, improve performance or make a static decision regarding the change. Given these constraints we defined the process to select the best arm as the node (h, i) with the largest criterion $C_{h,i}$, where

$$C_{h,i} = \begin{cases} \frac{\widehat{\mu_{h,i}}(n)}{\sqrt{\frac{2\ln n}{T_{h,i}(n)}} + v_1 \rho^h}, & \text{if } T_{h,i}(n) > 0 \\ 0, \text{ if } T_{h,i}(n) = 0 \end{cases} \tag{11}$$

  The idea behind this criterion is to select the node with the largest average while having the lowest bound. This penalizes nodes that have been play few times compared to nodes that have a higher confidence. A downside of this is that it favors nodes that had several plays, and this is usually associated with nodes at lower heights. However, this criterion performs better than the suggested highest height node, when measuring the distance to the theoretical best arm using an absolute Euclidian distance. Note that this criterion does not influence the growth of the tree as it still uses the upper confidence bound to select the best child node.
- The LG-HOO introduces a restriction to the height of the tree. As the tree grows it becomes computationally intensive to update all bounds in a single play iteration. Delays and computational restriction if running in computers with limited resources (such as embedding the algorithm in mobile apps) can significantly impact the user experience. Restricting the tree height puts an upper bound in the computational time, however it limits the precision that the algorithm can reach.
- To facilitates the understanding of the user behavior after running the algorithm for a limited time period, we make an estimation of the underlying mean-reward function using the Savitzky-Golay filter [26] with the decision criterion. Empirically, we determine that a window size of (number of nodes)/2 and a polynomial order equal to the tree's height, to produce good results. The tradeoff of using the Savitzky-Golay smoothing filter is the underestimation of high derivative peaks, leading to a conservative estimation.
- In practice, it often happens that an experiment is coded and then launched without being active (showing for all users the same variation). When the experiment is finally launched several users might be using arms that are not the defined root of the HOO algorithm. Similar situation can also happen in approximation of values by different users/clients. The LG-HOO tries to minimize the number of lost data points by selecting the closest node. If the played arm is closer to the node then its children, the reward is added to the node, otherwise it is discarded. This strategy works under the assumption of continuity of the underlying function while it minimizes the number of discarded data points.

Algorithm 1 represents the full LG-HOO strategy, using the same notation as the HOO, as discussed in the background. This algorithm is implemented in Python 2.7 and is available at https://github.com/davidissamattos/LG-HOO.

Algorithm 1 The LG-HOO algorithm

Input: $\nu_1 > 0$, $\rho \in (0,1)$, minimum growth $\gamma > 0$, tree maximum height σ
Initialization: $\tau = (0,1)$ and $B_{1,2} = B_{2,2} = +\infty$

procedure SELECT ARM(n)
  $(h,i) \leftarrow (0,1)$
  $P \leftarrow (h,i)$
  while $(h,i) \in \tau$ **do**
    if $B_{h+1,2i-1} > B_{h+1,2i+1}$ **then**
      $(h,i) \leftarrow (h+1, 2i-1)$
    else
      if $B_{h+1,2i-1} < B_{h+1,2i+1}$ **then** $B_{h+1,2i-1} > B_{h+1,2i}$
      else
        Z Ber(0.5)
        $(h,i) \leftarrow (h+1, 2i-Z)$
    $P \leftarrow P \cup (h,i)$
  $(H,I) \leftarrow (h,i)$
  Selected arm $\leftarrow U(1, P_{H,I})$
procedure UPDATE TREE(a_{played}, n, reward Y)
  Select the closest arm a_{played} to the node $v((H,I))$
  if $|a_{\text{played}} - v((H,I))| < \frac{|v((H+1,2I)) - v((H+1,2I-1))|}{2}$ **then**
    $a_{\text{played}} \leftarrow v((H,I))$
  else
    break
  for all node in $P_{H,I}$ **do**
    $T_{h,i} \leftarrow T_{h,i} + 1$
    $\mu\hat{}_{h,i} \leftarrow (1 - \frac{1}{T_{h,i}})\hat{\mu}_{h,i} + \frac{Y}{T_{h,i}}$
  for all node in τ **do**
    $U_{h,i} \leftarrow \hat{\mu}_{h,i} + \sqrt{(2 \ln n)/T_{h,i}} + \nu_1 \rho^h$
  if $T_{H,I} > \gamma$ **and** $H < \sigma$ **then**
    $B_{H+1,2I-1} \leftarrow +\infty$
    $B_{H+1,2I} \leftarrow +\infty$
  while node $\neq (0,)$ **do**
    $(h,i) \leftarrow$ new leaf
    $B_{h,i} \leftarrow \min\{U_{h,i}, \max(B_{h+1,2i-1}, B_{h+1,2i})\}$
procedure SELECT BEST ARM
  $a_{\text{best}} = \max_\tau (\frac{\hat{\mu}_{h,i}}{\sqrt{(2 \ln n)/T_{h,i}} + \nu_1 \rho^h})$
procedure APPROXIMATION OF THE UNDERLYING FUNCTION
  $\hat{f} =$ `Savitzky-Golay`$((\frac{\hat{\mu}_{h,i}}{\sqrt{(2 \ln n)/T_{h,i}} + \nu_1 \rho^h}),$ number of nodes$/2, \max_\tau h)$

The repository presents additional information on the connection of the algorithm with the implemented code. The algorithm is composed of four procedures. The first procedure is the procedure that selects the arm to be played in with the current tree. The second is called after an arm is played and a reward is received, updating and extending the tree. The third selects the best arm, when the optimization process is being finalized. The forth estimates the mean-reward function using the Savitzky-Golay filter.

4.1 The LG-HOO in Simulation

In this subsection, we provide some illustrative pictures of the LG-HOO algorithm in a simulation environment using different mean reward functions. Figure 1 shows the

usage of the LG-HOO algorithm in 6 different conditions. The orange line is the true mean-reward function that determines the probability of a Bernoulli distribution $Y = Ber(f(x))$. Where Y is the measure value (0 or 1, click or no click) and $f(x)$ is the mean reward function with variation x. This line can represent a customer profile (that is unknown but we still want to optimize a variation for this function). This profile can be complex as the picture in the left-top corner or simpler such as the picture in left-middle with only three ranges of value. The optimization process consists of finding the variation x that maximizes the mean reward function based only on the stochastic measured Y.

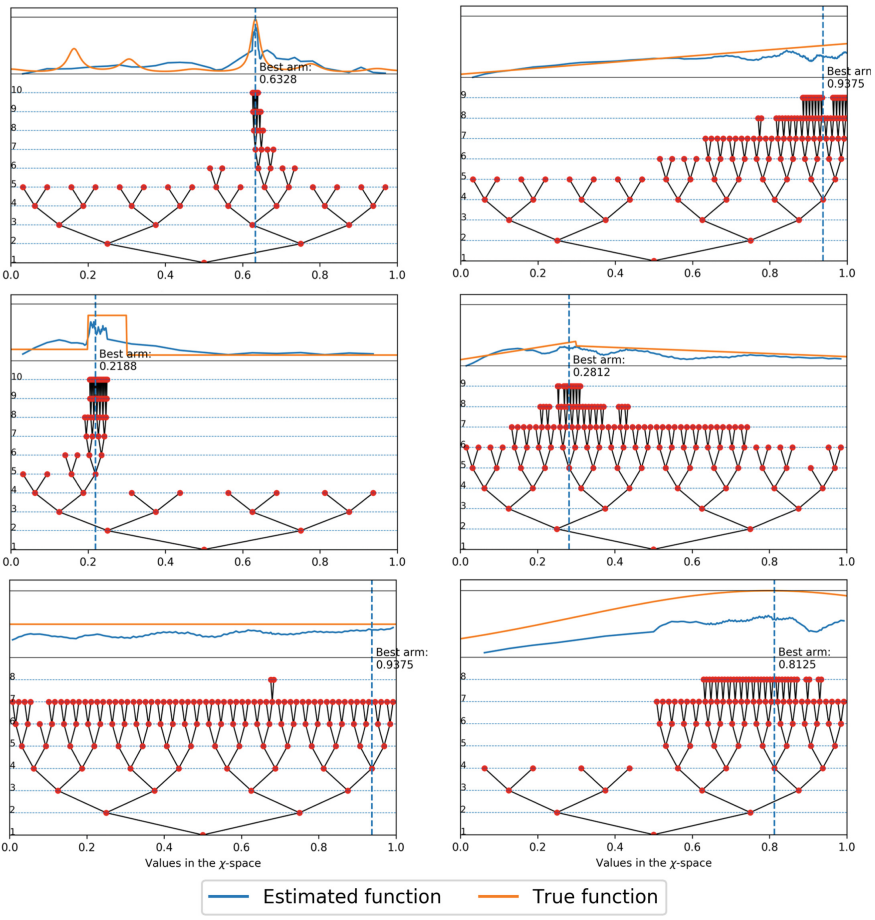

Fig. 1. Simulation results of the LG-HOO algorithm in wide range of different user mean-reward functions. In orange, is the true mean-reward function (unknown to the LG-HOO). In blue, is the estimated mean-reward function. The tree represents the LG-HOO search process at the end of the iteration, and the blue vertical line represents the best arm selection using the proposed selection criterion. The top-left subplot represents the same mean-reward function discussed in the original HOO algorithm [21]. (Color figure online)

These simulations show how the LG-HOO algorithm work and estimate the mean reward function (blue line). All the simulations were conducted considering a total of 10,000 unique interactions (horizon $n = 10,000$), using the minimum growth of 10, maximum tree height limit of 10, $v_1 = 1.0$, and $\rho = 0.5$, which is representative of the amount of data collected in a period of one month of the conducted experiment with Sony Mobile. We can see that with this number of unique interactions we can estimate the parameter that maximizes mean reward function.

4.2 The LG-HOO at Sony Mobile

The LG-HOO was implemented in the context of the product described in Sect. 3. One of the features of the product has an algorithm that estimates the time for launching a notification to users. If the notification arrives too early the users can ignore it and the feature has little value. If it arrives too late it can have a negative impact in the overall user experience. Before the experiment, the feature was using the minimum time scenario (reducing even more the time makes the notification arrives too late). The impact of the notification is measured depending on the action the user takes after receiving the notification. This metric is a stochastic variable that follows a Bernoulli distribution, where 1 (positive value) represents when the user takes an action in time and 0 when the user does not take an action in time (negative). The metric is stochastic because different factors not related to the time of the notification might influence the user action. The team wanted to investigate if a change in the algorithm that modifies the notification time impacts the metric. The hypothesis of this experiment is that adding a constant delay in the algorithm could indicate the extent the algorithm influences the metric and if development effort was needed to improve it. The team also wanted to minimize the regret of too early notifications. Sequential A/B/n experiments would take too long to cover the whole extent of search space while increasing the regret. This scenario sets an appropriate experiment for a continuum-armed bandit algorithm such as the LG-HOO.

The experiment consisted of searching an appropriate delay offset for the notification. The experiment limited the offset in the range of 0 and 600,000 ms (10 min). The users were assigned to a new variation delay every time they launched their mobile applications, and they logged their behavior right after the timeout to complete the action.

The LG-HOO was implemented in the ACE system in Python 2.7. The ACE system is hosted in the Google App Engine Flexible cloud environment[1]. The company application logged data and requested variation arms from the ACE system using POST requests. In case of lost packages or failure in requesting a new variation, the system uses the current variation (offset of zero). The parameters of the LG-HOO in this scenario are: minimum growth of 10, maximum tree height limit of 10, $v_1 = 1.0$, and $\rho = 0.5$. The limit in the tree height restricts the precision of the output of the algorithm in approximately 500 ms, which is considered good level of precision for the application. For this experiment, it was collected data from over 5000 user interactions

[1] https://cloud.google.com/appengine/docs/flexible/.

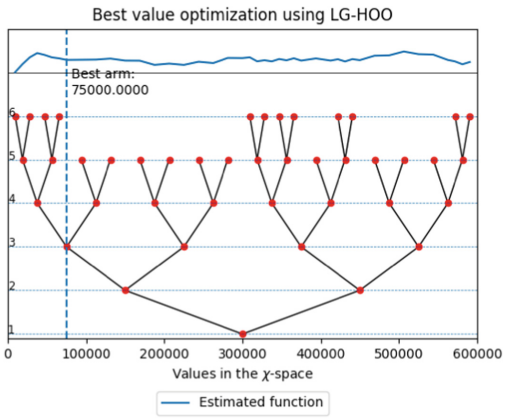

Fig. 2. The LG-HOO used in the Sony Mobile case. This picture provides both the visualization of the search tree, as well as the approximated mean-reward function and the selected best arm.

in the period of 4 weeks. The results and the outputs of the algorithm are shown in Fig. 2. This Figure provides both the visualization of the search tree, as well as the approximated mean-reward function and the selected best arm. The mean-reward function indicates that the offset does not have a large influence in the selected metric for the extent of the whole range of delays, but it still shows that a small delay can improve the concerned metric.

For the team the approximation of the of the mean reward function was important because it maps how the users behave in respect to this modification on the system, and therefore can help decisions such as to modify the feature, try other experiments on the feature or related features or move the development effort to another part of the system.

This section described the LG-HOO algorithm, the modifications, and trade-offs of the LG-HOO. In the simulation subsection, we provide simulation results and evidence of the LG-HOO being applied to different mean-reward functions. The simulation results allowed us to implement the LG-HOO algorithm with confidence in an industrial setting in collaboration with Sony Mobile. As there is no industrial evidence of the use of the HOO algorithm, some of its limitations were unknown prior to this work. The industrial case provides real-world evidence of the use of the LG-HOO in online experiments.

5 Discussion

Prior to launching the algorithm to real users, a comparison between the HOO and the LG-HOO was made and is discussed in this section. The algorithms were compared using the absolute Euclidian distance to the theoretical maximum and the time to compute an algorithm iteration. The first comparison looks at how far the algorithm got from the true value and relates to the following LG-HOO modifications: (1) the selection of the best arm policy modification and (2) the minimum number of times an

arm must be played before growing. The second comparison relates to degradation of user experience and performance of the system due to the introduction of delays in the estimation of the next arm to be played.

The algorithms were compared using a Monte Carlo simulation comparing one thousand runs with a horizon of $n = 1000$. At each simulation of the algorithm, it was used a generated random polynomial function as the true mean-reward function $f(x)$. The polynomial functions were generated by: (1) generating a set of 30 random points in the (x, y) plane, (2) fitting a polynomial with random order (ranging between 0 and 10) to these points, and (3) constraining both the space (x) and the mean reward probability (y) between 0 and 1. The user follows a Bernoulli distribution $Bern(f(x))$, where $y = f(x)$, and 1 represents a success. With this method we generate random polynomial functions that are used as the mean reward functions to simulate the user profile for the algorithms. With this method we can simulated both algorithms against the same set of true mean reward functions and compare the LG-HOO and the HOO solutions with the true solution using the absolute Euclidean distance.

The time spent in the calculation for selecting the next arm and the Euclidian distance were done in the same hardware and operational conditions. The data collected from this Monte Carlo simulation, and the conducted analysis is also available at repository. The collected data for the Euclidian distance and the time spent metrics for both algorithms are non-normal, Shapiro-Wilk test with $p < 2.2e{-}16$ and by visual inspection. Therefore, we compared the two algorithms metrics using the Mann-Whitney U non-parametric test [27]. We considered as null hypothesis that the respective LG-HOO metric does not differ from the HOO metric. Table 1 provides a summary of the statistical analysis. This statistical analysis provide evidence that the LG-HOO reduces the distance average Euclidian in 14.3% and reduces the spent time in 26%, using a confidence level of 95%. Due to the increased performance of the LG-HOO regarding to the correctness of the output and the computation time, only the LG-HOO algorithm was selected for empirical evaluation in the company case. The data and code to run this statistical analysis is available at the repository.

Table 1. Summary of the statistical analysis to compare the LG-HOO and the HOO algorithms using the Mann-Whitney U test, using a confidence level of 95%

Metric	Algorithm	Mean value	Absolute relative difference	P-value
Euclidian distance	LG-HOO	0.293	14.3%	0.04179
	HOO	0.335		
Time spent (in seconds)	LG-HOO	1.00	26%	<2.2e−16
	HOO	1.26		

6 Conclusion

Optimization procedures associated with bandit algorithms are of great interest to companies running online experiments. A particular case is the optimization of a continuous space in the presence of an unknown mean-reward function. As companies

develop their products, several user assumptions are incorporated into constants in their software development. Optimization in this scenario is a subclass of bandit problems called infinitely many-armed bandits. Previous research provides algorithms to solve this problem in the unidimensional space. However, these algorithms do not have empirical evidence or usage in online experiments and have restrictions that prevent their utilization as proposed. This work explores the unidimensional infinitely many-armed bandits problem in collaboration with Sony Mobile Communications.

The contribution of this work is three-fold. First, we present a modification of the Hierarchical Optimistic Optimization algorithm (HOO), called the Limited Growth Hierarchical Optimistic Optimization algorithm (LG-HOO). This modification is intended to overcome the problems associate with implementing the HOO algorithm in real-world online experiments. The modifications and the trade-offs involved with these modifications are presented. Second, the LG-HOO was implemented in collaboration with Sony Mobile. In this scenario, we provide real-world evidence of the usage of this algorithm for optimization of software constants. Third, we provide a statistical comparison between the LG-HOO and the HOO algorithm in simulation. The statistical analysis supports the conclusion that the LG-HOO perform better than the HOO, in the time spent to run and the accuracy of the results. These contributions support the relevance of the LG-HOO algorithm in the context of optimization experiments and show how the algorithm can be used to support continuous optimization of online systems in stochastic scenarios.

This work is the first step in analyzing the usage of infinitely many-armed bandit algorithms in optimization procedures in software development. In future work, we plan to expand the LG-HOO to support multi-dimensional arm space, support a multi-dimensional reward, as these are one of the key aspects that companies want to provide optimization, and validate these extensions in relevant industrial problems.

Acknowledgments. This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. The authors would also like to thank to all the support provided by the development team at Sony Mobile.

References

1. Kevic, K., Murphy, B., Williams, L., Beckmann, J.: Characterizing experimentation in continuous deployment: a case study on bing. In: Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017, pp. 123–132 (2017)
2. Fabijan, A., Dmitriev, P., Olsson, H.H., Bosch, J.: The evolution of continuous experimentation in software product development. In: Proceedings of the 39th International Conference on Software Engineering ICSE 2017 (2017)
3. Fabijan, A.: Developing the right features: the role and impact of customer and product data in software product development (2016)
4. Dmitriev, P., Wu, X.: Measuring metrics. In: Proceedings of the 25th ACM International Conference on Information and Knowledge. Management - CIKM 2016, pp. 429–437 (2016)

5. Schermann, G., Cito, J., Leitner, P.: Continuous experimentation - challenges, implementation techniques, and current research. IEEE Softw. **35**, 1 (2018)
6. Tang, D., Agarwal, A., O'Brien, D., Meyer, M.: Overlapping experiment infrastructure. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 2010, p. 17 (2010)
7. Bakshy, E., Eckles, D., Bernstein, M.S.: Designing and deploying online field experiments. In: Proceedings of the 23rd International Conference on World wide web - WWW 2014, pp. 283–292, September 2014
8. Kohavi, R., Deng, A., Longbotham, R., Xu, Y.: Seven rules of thumb for web site experimenters. In: Proceedings of the 20th ACM SIGKDD International Conference Knowledge Discovery and data Mining, KDD 2014, pp. 1857–1866 (2014)
9. Xu, Y., Duan, W., Huang, S.: SQR: balancing speed, quality and risk in online experiments, no. 1, pp. 1–9, January 2018
10. Fabijan, A., Dmitriev, P., Olsson, H.H., Bosch, J.: The benefits of controlled experimentation at scale. In: Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017, pp. 18–26 (2017)
11. Gui, H., Xu, Y., Bhasin, A., Han, J.: Network A/B testing. In: Proceedings of the 24th International Conference on World Wide Web - WWW 2015, pp. 399–409 (2015)
12. Li, L., Chu, W., Langford, J., Schapire, R.E.: A contextual-bandit approach to personalized news article recommendation. In: WWW 2010, p. 10 (2010)
13. Bottou, L., Peters, J., Quiñonero-Candela, J., Charles, D.X., Chickering, D.M., Portugaly, E., Ray, D., Simard, P., Snelson, E.: Counterfactual reasoning and learning systems. J. Mach. Learn. Res. **14**, 3207–3260 (2013)
14. Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google vizier. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 2017, pp. 1487–1495 (2017)
15. Tamburrelli, G., Margara, A.: Towards automated A/B testing. In: Le Goues, C., Yoo, S. (eds.) SSBSE 2014. LNCS, vol. 8636, pp. 184–198. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09940-8_13
16. Sutton, R.S., Barto, A.G.: Sutton & Barto Book: Reinforcement Learning: An Introduction. The MIT Press, Cambridge (1998)
17. Burtini, G., Loeppky, J., Lawrence, R.: A survey of online experiment design with the stochastic multi-armed bandit, pp. 1–49, October 2015
18. Shang, X., Kaufmann, E., Valko, M.: Hierarchical Bandits for "Black Box " Optimization, Lille, (2015)
19. Kohavi, R., Longbotham, R., Sommerfield, D., Henne, R.M.: Controlled experiments on the web: survey and practical guide. Data Min. Knowl. Discov. **18**(1), 140–181 (2009)
20. Wang, Y., Audibert, J.-Y., Munos, R.: Algorithms for infinitely many-armed bandits. In: Advances in Neural Information Processing Systems, pp. 1–8 (2008)
21. Bubeck, S., Munos, R., Stoltz, G., Szepesvári, C.: X - Armed Bandits. J. Mach. Learn. Res. **12**, 1655–1695 (2011)
22. Urban, G.L., Liberali, G.G., MacDonald, E., Bordley, R., Hauser, J.R.: Morphing banner advertising. Mark. Sci. **33**(1), 27–46 (2014)
23. Li, L., Chu, W., Langford, J., Schapire, R.E.: A contextual-bandit approach to personalized news article recommendation. In: Proceedings of the 19th International Conference on World Wide Web, 2010, pp. 661–670 (2010)
24. Mattos, D.I., Bosch, J., Olsson, H.H.: Your system gets better every day you use it: towards automated continuous experimentation. In: Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (2017)