Empirical Software Engineering (2020) 25:3841–3872
https://doi.org/10.1007/s10664-020-09856-1

Data-driven software design with Constraint Oriented Multi-variate Bandit Optimization (COMBO)

Rasmus Ros[1] · **Mikael Hammar**[2]

Published online: 18 August 2020

Abstract

Context Software design in e-commerce can be improved with user data through controlled experiments (i.e. A/B tests) to better meet user needs. Machine learning-based algorithmic optimization techniques extends the approach to large number of variables to personalize software to different user needs. So far the optimization techniques has only been applied to optimize software of low complexity, such as colors and wordings of text.

Objective In this paper, we introduce the COMBO toolkit with capability to model optimization variables and their relationship constraints specified through an embedded domain-specific language. The toolkit generates personalized software configurations for users as they arrive in the system, and the configurations improve over time in in relation to some given metric. COMBO has several implementations of machine learning algorithms and constraint solvers to optimize the model with user data by software developers without deep optimization knowledge.

Method The toolkit was validated in a proof-of-concept by implementing two features that are relevant to Apptus, an e-commerce company that develops algorithms for web shops. The algorithmic performance was evaluated in simulations with realistic historic user data.

Results The validation shows that the toolkit approach can model and improve relatively complex features with many types of variables and constraints, without causing noticeable delays for users.

Conclusions We show that modeling software hierarchies in a formal model facilitates algorithmic optimization of more complex software. In this way, using COMBO, developers can make data-driven and personalized software products.

Keywords Continuous experimentation · A/B testing · Machine learning · Multi-armed bandits · Combinatorial optimization

This article belongs to the Topical Collection: *Software Engineering in the Age of Artificial Intelligence*

Communicated by: Tim Menzies, Chakkrit Tantithamthavorn and Burak Turhan

✉ Rasmus Ros
rasmus.ros@cs.lth.se

Extended author information available on the last page of the article.

1 Introduction

Design of user-facing software involve many decisions that can be optimized with user data. The decision variables—called the *search space*—can include both product aspects that are directly or indirectly visible to the user. For example, what wordings to use in headings or how items should be ranked in recommender systems (Amatriain 2013) or search engines (Tang et al. 2010). Traditionally, randomized controlled experiments (i.e., A/B tests or split tests) are used to iteratively validate the design choices based on user data (Kohavi et al. 2008). Recently, data-driven optimization algorithms have been proposed to perform automated experimentation on software in larger scale on bigger search spaces simultaneously, at e.g., Amazon (Hill et al. 2017) and Sentient (Miikkulainen et al. 2017). Personalization in particular is touted (Hill et al. 2017) as an opportunity to apply optimization algorithms to improve the user experience for different circumstances in, e.g. device types or countries.

The benefits of using optimization algorithms need to be balanced against the cost of implementing it. If the implementation cannot be broadly applied to many parts of the software product this investment might not pay dividends. Previously, data-driven optimization algorithms have only been applied to simple software (Hill et al. 2017; Miikkulainen et al. 2018) with a flat structure in the decision variables, such as colors, layouts, texts, and so on. Software with more complex behaviours cannot be directly optimized with these techniques. We hypothesize that to handle more types of software the algorithms must understand the hierarchies that software is build with. For example, a software feature can have dependencies between variables such that one variable can only enabled if another one is, and so on.

We suggest modeling the search space and the relationships between variables—called *constraints* (Biere et al. 2009; Rossi et al. 2006)—in a formal language that can describe the software hierarchy. Developers can use constraints to exclude certain combinations from the optimization search space that would otherwise generate undesirable or infeasible variants. For example, the color of a button should not be the same as the background. Feature models (Chen et al. 2009; Kang et al. 2002) from software product lines have been suggested by Cámara and Kobsa (2009) as a suitable modeling representation to handle the variability of experimentation. With feature models, software variable dependencies are described in a tree hierarchy. Feature models also usually support a limited set of more complex constraints.

To this end we introduce the open-source toolkit called Constraint Oriented Multi-variate Bandit Optimization (COMBO) targeted at software engineers without deep optimization expertise. The toolkit consists of a domain-specific language for specifying a hierarchical model of the optimization search space with many types of constraints and multiple bandit-based machine learning algorithms (see Section 3) that can be applied to optimize a software product. To the best of our knowledge, this is the first attempt at combining bandit-based optimization algorithms with constraints. We have validated the toolkit's capabilities in a proof-of-concept by implementing two feature cases relevant to the e-commerce validation company Apptus. The algorithmic performance has been evaluated in simulations with realistic data for the feature cases.

Finally, we discuss the implications of using toolkits such as COMBO in the context of a data-driven software development process, which we define as the practice of *continuous optimization*. The current barriers to applying continuous optimization need to be lowered in order to encourage and enable developers to shift towards a higher degree of experimentation. For example, since modern software products are in a state of constant change, the

optimization search space will have underperforming variables removed and new variables added. The algorithms need to gracefully handle such continuous updates of the search space model without restarting the optimization. We also call attention to several remaining barriers such as: handling concept drift (Kanoun and van der Schaar 2015) and ramifications on software testing (Masuda et al. 2018). We also provide considerations for what metrics could be optimized for and what the toolkit could be applied to.

The rest of this paper is structured as follows. Section 2 contains background and related work on continuous experimentation. Section 3 introduces theory on bandit optimization. In Section 4 the research context and methods are described along with threats to validity of the validation and limitations of the solution. In Section 5 the COMBO toolkit is presented. In Section 6 the toolkit is validated and the algorithms are evaluated. Finally, Sections 7 and 8 discuss continuous optimization, metrics, future directions, and conclude the paper.

2 Background and Related Work on Continuous Experimentation

Many web-facing companies use continuous experimentation (Fagerholm et al. 2017) for gauging user perception of software changes (Auer and Felderer 2018; Ros and Runeson 2018). By getting continuous feedback from users software can evolve to meet market needs. Randomized controlled experiments (i.e. A/B tests, split tests, etc.) in particular are emphasized by high-profile companies such as Microsoft (Kevic et al. 2017), Google (Tang et al. 2010), and Facebook (Feitelson et al. 2013) as an evidence-based way of designing software. The section below provides background on continuous experimentation through the lens of randomized controlled experiments for software optimization. This gives context to the main topic of our work on applying data-driven software optimization algorithms.

Experiments can be executed either on unreleased prototypes or after deployment to real users. Bosch-Sijtsema and Bosch (2015) and Yaman et al. (2016) explain how qualitative experiments on prototypes are used early in development to validate overarching assumptions about user experience and the business model. While post-deployment experiments, such as randomized controlled experiments, are used to measure small differences between software variants for optimizing a business related or user experience (UX) metric.

Prototype experiments are advocated for both in the lean startup framework by Ries (2011) and in user experience research (Chamberlain et al. 2006; Williams 2009). Lean startup is about finding a viable business model and product through experimentation, for example, a pricing strategy suitable for the market. Lean startup has been brought to software product development in the RIGHT model by Fagerholm et al. (2017). Experiments based on design sketches are used within user experience to validate user interaction design, e.g. through user observations.

In both lean startup and user experience research there is a need to get feedback on prototypes early in the process. Though, as the product or feature design matures and settles, the shift can move towards optimization with randomized controlled experiments to fine tune the user experience. This can exist simultaneously with prototype experiments as different features in a product can have different levels of design maturity.

2.1 Randomized Controlled Experiments

In a *randomized controlled experiment*, variables are systematically changed to isolate the effect that each setting of the variable has on a certain metric. The variable settings are

mapped to software configurations and each unique software configuration is assigned a user experiment group. When the experiment is deployed to a product environment each user of the system is randomly assigned to a user experiment group. Usually there are thousands of users per group to obtain statistical significance.

Randomized controlled experiments have been studied in data science as *online controlled experiments*. The tutorial by Kohavi et al. (2008) from Microsoft provides a good introduction to the statistics and technicalities of it. The research includes studies on e.g. increasing the efficiency (Fabijan et al. 2018a) and trustworthiness of results (Kohavi et al. 2012).

The structure of the controlled experiment is referred to as the experiment design. In an A/B test there are two user experiment groups and in an A/B/n test there can be any number, but still with one variable changed. Thus, they are univariate. In a multi-variate test (MVT) there are also multiple variables that each can have multiple values. There are different strategies for creating experiment groups in an MVT. For example, in the full factorial design all interaction terms are considered so if there are n binary variables there would be 2^n groups. In a fractional factorial design only some interactions are considered.

Infrastructure is a prerequisite for controlled experiments on software (Fagerholm et al. 2017). The bare minimum is an experimentation platform that handle the randomized assignments of users and statistics calculations of the experiment groups. Microsoft have described their experimentation platform (ExP) for conducting experiments in large scale (Gupta et al. 2018). It has some additional optional features such as segmentation of users with stratified sampling for cohort analysis, integration with deployment and rollback of software, sophisticated alerting of suspected errors, and so on.

2.2 Personalization

MVTs are the current standard experiment design for having personalized experiment results (Hill et al. 2017; Kohavi et al. 2008). By *personalization* (Felfernig et al. 2010; Schwabe et al. 2002) we mean that there are contextual variables that describe aspects of a user, such as device type or age, and that the point of personalization is to find different solutions for the different combinations of contextual variables. Having many personalization variables will result in needing many more experiment groups.

In the classical experiment designs of A/B/n tests and MVTs, users are allocated into experiment groups uniformly at random. That is, each experiment group will have equally many users. When the number of groups is large this can be inefficient. In any optimization algorithm, the allocation of users to experiment groups changes based on how well it performs in relation to some metric. Thus, they can concentrate users to the most promising configurations.

2.3 Experimentation Implementation Strategies

There are two distinct implementation strategies for randomized controlled experiments of software: using feature flags or multiple deployment environments. Firstly, feature flags (Rahman et al. 2016) are essentially an if/else construct that toggle a feature at run time. This can be extended to do A/B testing. Secondly, having multiple software releases deployed to different environments, ideally done through containerized blue-green deployment (Révész and Pataki 2017). The advantage of this approach is that the software variants can be organized in different code branches.

The number of experiment groups can be huge, especially with personalization and optimization. For the algorithmic optimization advocated in this work, having deployment environments for each combination of variable settings is infeasible. Thus, the feature flag strategy is presumed. However, there are scheduling tools that optimize the efficiency of experimentation in different deployment environments by Schermann and Leitner (2018) and Kharitonov et al. (2015).

2.4 Model-Based Experimentation and Variability Management

Experimentation introduces additional variability in software by design. Cámara and Kobsa (2009) suggested modeling the variables through a *feature model* from software product lines research (Kang et al. 2002). Feature models allow for the specification of hierarchies and other relations between feature flags and have been used to capture variability in many software products (Chen et al. 2009). With feature models one can perform formal verification on the internal consistency of the model and perform standard transformations. This approach does not seem to have gained traction for continuous experimentation. Our approach of adding constraints and hierarchies (see Section 5) is not new per se, but the use of this in combination with optimization is novel to the best of our knowledge.

In practice, less formal methods are used for configuring many overlapping experiments at Google (Tang et al. 2010) and Facebook (Bakshy et al. 2014). Facebook has open sourced parts of their infrastructure for this in the form of a tool (Bakshy et al. 2014) (PlanOut) that can be used to specify experiment designs. The tool also contains a namespace system for configuring overlapping experiments. In both companies' approaches, they have mutual exclusivity constraints where each experiment claims resources, for example, a specific button on a page. A scheduler or other mechanism ensures that experiments can run in parallel without interfering with each others resources.

2.5 Automated Experimentation with Optimization Algorithms

There is an abundance of tools that optimize parameters in software engineering and related fields, e.g., tweaking machine learning hyper-parameters (Borg 2016; Snoek et al. 2012), finding optimal configurations for performance and efficiency (Hutter et al. 2014; Hoos 2012; Nair et al. 2018), tweaking search-based software engineering parameters (Arcuri and Fraser 2011), and the topic of this work with software parameters for business metrics. In the various optimization applications, the assumptions are different on how the optimization problem is structured and what the technical priorities are. For example, when optimizing machine learning hyper-parameters for deep learning it is important to minimize the number of costly experiments. Bayesian optimization with Gaussian processes (Snoek et al. 2012) is often used there, but the approach does not scale beyond a few thousand data points (Riquelme et al. 2018), because the computational cost depends on the number of data points for Gaussian processes.

One notable related research field is autonomic computating (Kephart and Chess 2003) that includes self-optimization of performance and efficiency related parameters, such as the loading factor of a hash table or what implementation to use for an abstract data structure. Such performance factors have relatively high signal-to-noise ratio in comparison to metrics involving users and the factors also exhibit strong interactions in terms of memory and CPU trade-offs. Many of these optimization tools (e.g. in Hoos 2012; Nair et al. 2018) assume that the optimization is done before deployment during compilation in a controlled

environment. Some recent work has moved the optimization to run time and studied the required software architecture and infrastructure for cyber-physical systems (Jiménez et al. 2019; Gerostathopoulos et al. 2018) and implications (Mattos et al. 2017; Gerostathopoulos et al. 2018) of this change.

We have also found several optimization approaches similar to our work, viz., they target large search spaces, with metrics based on many users, and are applied at run time in a dynamic production environment. The most similar work to ours is by Hill et al. (2017) at Amazon where the problem formulation of multi-variate multi-armed bandits (see next section) is first formulated and addressed with a solution. There is also related work on search-based methods (Iitsuka and Matsuo 2015; Miikkulainen et al. 2017; Tamburrelli and Margara 2014) and hybrids between search-based and bandit optimization (Miikkulainen et al. 2018; Ros et al. 2017).

Search-based tools was first suggested by Tamburrelli and Margara (2014) for automated A/B testing using genetic algorithms, and then independently by Iitsuka and Matsuo (2015) for website optimization using local search. Ros et al. (2017) suggested improving the genetic algorithms with bandit optimization and a steady state population replacement strategy. At Sentient they have implemented some of these ideas (Tamburrelli and Margara 2014; Ros et al. 2017) in a commercial tool with both genetic algorithms (Miikkulainen et al. 2017) and genetic algorithms with bandit optimization (Miikkulainen et al. 2018).

A genetic algorithm maintain a population of configurations and evaluate them in batches. The configurations are ranked by a selection procedure, and those that perform well are recombined with each other with some probability of additional mutation. A genetic algorithm were implemented in our toolkit. However, genetic algorithms were not investigated further in our work because we are not aware of an elegant way of doing personalization with genetic algorithms. Maintaining a separate population for each combination of context variables is not feasible, because each separate population would have too few users. What we implemented in the toolkit was that when a user arrives with a given context, try to match it with a configuration in the algorithm's population, if there is no match then generate a new configuration with the selection procedure that is coerced to match the context and add it the population. That scales better to more context variables due to concentrating effort to popular contexts, it eventually runs into the same problem with too few users per configuration.

3 Theory on Bandit Optimization of Software

Bandit optimization is a class of flexible optimization methods, see Fig. 1 for a summary. Univariate bandit optimization is formalized in the multi-armed bandit problem (MAB) (Burtini et al. 2015). The name come from the colloquial term one-armed bandit which means slot machine. In MAB, a gambler is faced with a number of slot machines with unknown reward distributions. The gambler should maximize the rewards by iteratively pulling the arm of one machine. Applying MAB to A/B/n testing is sufficiently common to have its own name: bandit testing (such as in Google Optimize). The choice of arm is sometimes called an action but is referred to as configuration in this work.

An optimizer policy solves the MAB problem. Some of the policies are very simple, such as the popular ϵ-greedy. It selects a random configuration with probability ϵ, otherwise the configuration with the highest mean reward. A policy that performs well must explore the optimization search space sufficiently to find the best configuration. Policies must also

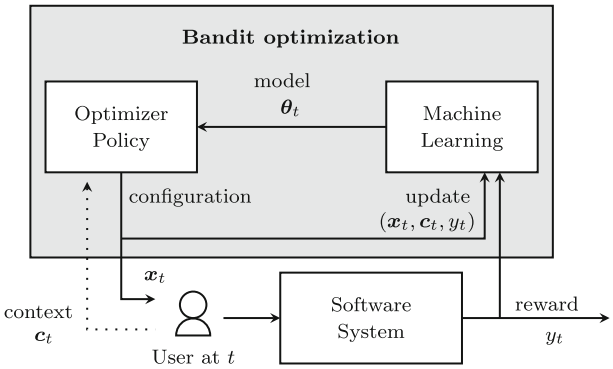


Fig. 1 Bandit optimization setting summary. A user at time t of the system provides context $\boldsymbol{c}_t$ and receives personalized configuration $\boldsymbol{x}_t$. The user provides the reward y_t by using the software system. An optimizer policy selects configurations which maximizes rewards based on a machine learning algorithm model $\boldsymbol{\theta}_t$. The model can predict rewards based on configurations and contexts and is continuously updated

exploit the best configurations that it has found to get high rewards. This is known as the exploration-exploitation trade-off dilemma (Sutton et al. 1998) in reinforcement learning. In ϵ-greedy this trade off is expressed explicitly in the ϵ parameter. A high ϵ results in random exploration, and low results in ϵ in pure exploitation of the best configuration and runs the risk of getting stuck with a local optima.

In the multi-variate case, the MAB setting is extended by a machine learning algorithm that attributes each reward to the variables in the multi-variate arm. This is known as multi-variate MAB (Hill et al. 2017) or more generalized as combinatorial MAB (Chen et al. 2013). Contextual MAB is another more well known extension used for e.g. recommender systems (Li et al. 2010) which does personalization in the univariate case. In this work, we refer to all of these settings as bandit optimization, but the focus is on multi-variate MAB.

Herein lies the crucial difference between the multi-variate MAB setting and other forms of black-box optimization (where the objective function is unknown). Namely, that the algorithm learns a *representation* in the form of an online machine learning model and optimizes it with respect to its inputs. Some black-box optimization algorithms (Arcuri and Fraser 2011; Hoos 2012; Nair et al. 2018) find and keep track of the best performing data points and iteratively replaces them with better performing ones. That is not the case in our work because of the assumption that the signal-to-noise ratio is so low that estimating the performance of a specific data point is inefficient. Other black-box optimization algorithms (Hutter et al. 2014; Riquelme et al. 2018; Snoek et al. 2012) have offline machine learning models that need to keep track of all data points and retrain the machine learning model at intervals, that does not scale well to very large data sets.

To summarize so far, a more precise definition of bandit optimization follows. A user arrives in the system at time step t. A software configuration $\boldsymbol{x}_t \in \mathcal{X}, x_{t,i} \in \mathbb{R}, i = 1, \ldots, n$, is chosen for the user with context variables $\boldsymbol{c}_t \in \mathcal{C}, c_{t,j} \in \mathbb{R}, j = 1, \ldots, m$, where n is the number of configuration variables, m the number of context variables, $\mathcal{X}$ is the search space of decision variables, and $\mathcal{C}$ is the context space. When the user is no longer using the software system, a reward $y_t \in \mathbb{R}$ in some metric is obtained from the user. A policy is tasked with selecting $\boldsymbol{x}_t$ such that the rewards are maximized over an infinite time horizon.

3.1 Practical Implications on Software Systems

When applying bandit optimization in practice there are multiple assumptions:

- The number of users are at least measured in the thousands. Depending on how many variables are included in the search space and the signal-to-noise, the required number of users will be more or less.
- The value incurred by a user must be quantified to a single value at a discrete time. This will be a simplification because users continuously use software.
- The optimization is done online at run time with continuous updates. A user's configuration must be generated quickly, so that users do not suffer noticeable delays. For software with an installation procedure (desktop or mobile) the optimization can be done during installation.
- The rewards are assumed to be identically and independently distributed for all users. The algorithms are not *guaranteed* to perform well when the reward distributions change over time, though they still might perform well.

Additionally, in comparison to A/B testing, bandit optimization requires storing the configuration per user. Standard procedures for controlled experimentation assign experiment groups with a pseudo-random allocation with a random seed based on something that is static and unique for the user, e.g., a user's id. As such, a user will have the same experiment group even if their session expires. With bandit optimization, they might be assigned to another group unless the group assignment is stored persistently. Persistant storage of user group assignment has implications on data privacy (Hadar et al. 2018).

Mattos et al. (2019) conducted an empirical investigation into bandit testing for software. They found that while the technique performs better at finding good configurations, it can lead to statistical issues such as inflated false positive error rates. They advice to apply bandit testing only when the consequences of false positives are low, as it is with optimization.

3.2 Thompson Sampling

The optimizer policy of choice in our work is Thompson sampling since it often performs better than ϵ-greedy in general (Chapelle and Li 2011; Russo et al. 2017). The idea is to match the probability of selecting an arm with the probability of it being optimal. It works as follows for standard MAB. A Bayesian posterior distribution is placed on the expected reward of each arm. In each step t, a sample $\hat{\theta}_{k,t}$ is drawn from each posterior distribution $P(\theta_k|\boldsymbol{y}_k)$, where k is an arm index, θ_k is the prior reward parameter of the respective arm, and $\boldsymbol{y}_k$ its previous rewards. The arm a_t at step t with the highest sample is selected, that is $a_t = \arg\max_k \hat{\theta}_{k,t}$. The posterior distribution is updated continuously.

For example, if the rewards are whether users click on something or not. Then, a suitable posterior distribution family for binary rewards is the Beta-distribution, parameterized by the number of clicks and non-clicks. Figure 2 shows a simulation of this, comparing Thompson sampling, ϵ-greedy with $\epsilon = 0.2$, and random (as in standard A/B/n testing). Thompson sampling and ϵ-greedy initially behave like the random policy, but improve quickly. Thompson sampling finds the optimal arm, while ϵ-greedy is stuck with a random arm with probability 0.2.

For multi-varate MAB the posterior distribution is a probabilistic machine learning method denoted q parameterized by the machine learning model $\boldsymbol{\theta}$. A single joint sample $\hat{\boldsymbol{\theta}}_t$ is drawn from the multi-variate distribution $P(\boldsymbol{\theta}|\boldsymbol{y})$ each step. The configuration $\boldsymbol{x}_t$ is then

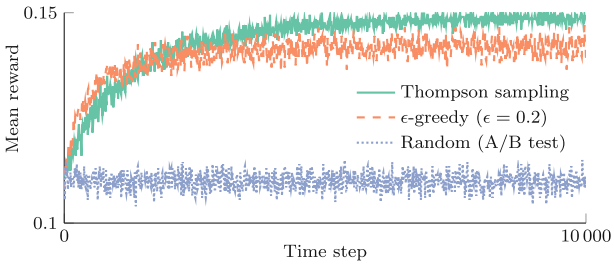


Fig. 2 Example simulation of different bandit policies with binary rewards. There are five available arms, four with mean reward 0.1 and one with reward 0.15. The simulation is repeated 100 times and the plot shows the average result. The figure illustrates that Thompson sampling eventually converges to the optimal reward while ϵ-greedy converges to $0.1\epsilon + 0.15(1-\epsilon) = 0.14$

chosen as such: $\boldsymbol{x}_t = \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathbb{E}_{q_{\hat{\theta}}}[y_t|\boldsymbol{x_t} = \boldsymbol{x}, \boldsymbol{c}_t]$. Depending on the machine learning method q this optimization problem is set up differently and it will be significantly harder to solve than in the univariate case. In MAB, the computational complexity at each step is linear in the number of arms, the arg max calculation can be a for-loop over each sample. While for multi-variate MAB, the computational complexity at each step is NP-hard. Section 5.2 contains the specifics for how we did the setup of the optimization for different machine learning methods in the toolkit.

4 Research Context and Methods

Our research was an on-site collaboration with the e-commerce company Apptus spanning 20 weeks, and was part of a long term research project. The research was conducted with a design science methodology, following the guidelines by Wieringa (2014) and Runeson et al. (2020). As such, the toolkit was designed as a solution that addresses an industrially relevant problem, and the validation of the solution is done with sufficient rigor (see Fig. 3 for an overview). This section ends with discussing the limitations of the solution design and threats to validity of the validation.

4.1 Validation Company and e-Commerce

The validation company Apptus is a small Swedish company which develops a platform for e-commerce. Their product platform provides web shops with various data-driven

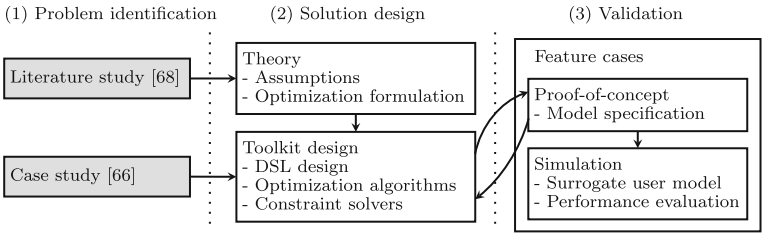


Fig. 3 Research methods overview divided in three stages. Based on previous studies (gray boxes), the theory was identified and the design of the toolkit was designed to support the validation company's product. The toolkit was evaluated with two feature cases, that were first implemented in the toolkit and then subject to simulations

algorithms. It includes a recommender system, product search engine, ads display algorithms, etc. Apptus deploy their software to other companies' web shops, so they have a business-to-business relationship (B2B) with their customers. Apptus has no direct relationship with the end-users of the software (consumers), but have access to consumer data through their customers' web shops. Operating multiple web shops incur a greater need for personalization (see Section 2.2) to optimize their software for different circumstances.

Experimentation is well established in e-commerce for a number of reasons; we believe this is the case for four reasons. First, the consumers often have a clear goal in mind to purchase a product. Second, this goal is aligned with the goals of the web shop companies. Third, there is an industry standard for quantifying this joint goal through the sales funnel of: clicks, add-to-carts, and purchases. Finally, consumers tolerate some degree of change in the interface, especially in what ads and products are displayed.

A prior case study by Ros and Bjarnason (2018) outlines how Apptus uses continuous experimentation to improve their product in several scenarios: validating that a change has the intended outcome, manual optimization, and algorithmic optimization. Thus, Apptus is experienced with using optimization in various forms to optimize the web shops. They also use bandit optimization in their product recommendation system (Brodén et al. 2017) and as part of a customer facing experimentation platform (targeted at marketers and software engineers) that optimizes which algorithm should be active in which parts of the web shop.

4.2 Research Stages

The design and validation of the COMBO toolkit was done in three stages (see Fig. 3): (1) identifying a problem with industrial relevance through a literature study (Ros and Runeson 2018) and a case study with interviews (Ros and Bjarnason 2018), (2) designing a solution to solve the problem, and (3) validating that the solution works.

4.2.1 Problem Identification

Prior to this study, two studies were performed for problem identification. First, a systematic mapping study (Ros and Runeson 2018) on continuous experimentation identified suitable algorithms to the problem domain and the assumptions in the optimization formulation as stated in Section 3. Second, an interview study with five participants from the validation company was conducted (Ros and Bjarnason 2018). The interviews investigated the difference between experimentation and optimization approaches, such as finding benefits and challenges of the respective approaches. One prominent challenge was that optimization algorithms are specific to a certain circumstance (e.g. product recommendations) and are hard to apply outside these circumstances. This challenge was also present in related work in their narrow application to visual design and layout.

4.2.2 Solution Design

The design of the toolkit was done in iterations with subsequent validation, to ensure that the toolkit had the necessary functionality to support the validation. The decisions taken in the design include what optimization algorithms to implement, what constraints to support, and how the search space should be specified. The design choices were anchored in two workshops with employees at the validation company. The approach was to be inclusive in terms of optimization algorithms by using available optimization libraries. The constraints supported were simply added as needed. The search space and constraints specification was

a choice between meta-programming with annotations that some related work used (Hoos 2012; Tamburrelli and Margara 2014) and declarative code with a domain-specific language, where the latter was chosen because it is more flexible for users of the toolkit.

4.2.3 Validation

The toolkit was validated on-site at the validation company by two *feature cases*. Each feature case include a validation proof-of-concept demonstration and evaluation simulations. Technical details on the simulation set up is given in Section 6 alongside the presentation of the feature cases. The first feature case, an auto complete widget, was identified in a brainstorming workshop with three employees from the validation company. It was chosen because it has both a graphical interface and is a data-driven component while being a well isolated part of the site. The second feature case, a top-*k* categories listing, was selected in a discussion with an employee to push the boundaries of what is technically feasible with the toolkit and because it was an existing feature that had real historic user data.

The proof-of-concept validation was conducted to demonstrate the toolkit's soundness. It included a step to ensure that all necessary variability of the feature case was captured and then demonstrating that it could be implemented in the toolkit. For the auto complete feature we found a listing of top 50 fashion sites and filtered it down to 37 fashion web shops that had an auto complete widget. We analyzed how the widgets varied and then implemented variables and constraints to sufficiently capture the variability. In addition, there was a brainstorming workshop with a user experience designer to validate the choices. For top-*k* we chose a client web shop that had a sufficient number of users and a complex category tree and re-implemented the functionality of the original top-*k* categories.

Simulations of the feature cases were performed to show that the optimization algorithms can navigate the search space in ideal circumstances, within a reasonable data and execution time budget. The use of simulation avoids the complexities of a deployment and the evaluation can be repeated and reproduced as many times as needed. Also, it enables benchmarking the different algorithms. In the simulation, the optimization algorithm iteratively chooses a configuration from the search space and a reward from the user is simulated to update the algorithm. However, the historic user data could not be used directly to simulate users, because all combinations of the variables in the model are not present in the data. Instead we used the historic data to train a supervised machine learning model that can predict the expected reward of each configuration. This prediction can be used to simulate user rewards from any configuration. The machine learning model is an instance of a *surrogate model* (Forrester et al. 2008), which is a general engineering technique used as a replacement for something that cannot be directly measured, which are users in this case.

4.3 Data Collection

Both qualitative research notes and quantitative data sets for evaluation were collected. Decisions taken during solution design and implementation of the feature cases were recorded as notes, as recommended by Singer et al. (2008). The notes were consulted during the write up. The notes were of two types. First, in the form of 20 weekly diaries when on-site at the validation company. They contained notes on decisions taken and considered during solution design, and implementation of the feature cases. Second, notes were taken after the three workshops outlined above, (1) when identifying the first feature case, (2) when presenting the toolkit design choices, and (3) when brainstorming the variability of the first feature case.

The data sets used to train the surrogate user model were collected at the validation company. The second feature case (the top-*k* categories) had historic data that was used. For the first feature case (the auto complete widget) we collected data through a questionnaire at the validation company. There we collected screenshots from the 37 fashion web sites with an auto complete widget. Then 16 employees were asked to rate 20 randomly chosen screenshots on a 10 point Likert scale to rate the user experience of the auto auto complete widget. Many of them mentioned that it was hard and somewhat arbitrary. However, the data set was only used to get a somewhat realistic benchmark, in comparison to a completely synthetic benchmark with randomly generated data.

4.4 Limitations

There are technical and statistical limitations to what is possible with the toolkit. The number of variables that can be solved for within a given millisecond budget is limited by the number of constraints, how sparse the configuration is (i.e. how many variables are zero valued), and the complexity of the underlying machine learning model. The ability to make inference from the learned model will also depend on the machine learning model complexity, for example, inference from neural networks is notoriously difficult. This limitation is a fundamental trade-off between algorithmic performance and inference ability.

There are also limitations on the impact that can be obtained from applying the optimization. In e-commerce, much of the interface can be measured directly in terms of its impact on revenue, thus the ability to have an effect is promising. We believe the toolkit can be of use also outside of e-commerce, although there must be a way to quantify the user experience of a given feature.

Finally, we are not claiming that all experiments in software engineering should be replaced by optimization. For instance, when adding optimization capabilities to a software feature, it would be prudent to validate that the optimization actually works with an ordinary controlled experiment.

4.5 Threats to Validity

Here threats to validity that threaten the validation are identified along with the steps taken to mitigate them.

The *external validity* of the feature cases are threatened by that the evaluation is performed with only one company. This is mitigated by that the validation company has multiple clients that operate web shops. Thus, the feature case implementations are relevant to multiple company contexts, still within the e-commerce domain though.

The *internal validity* of the validation is dependent on the quality of the datasets and the resulting surrogate user models. If the surrogate user models are too easily optimized by an algorithm the simulations will be unrealistic. For example, if there are regions in the optimization search space with low data support the surrogate user model might respond with a high reward there. To mitigate this threat we evaluated the performance with a baseline random algorithm and an oracle algorithm with access to the surrogate user model. In this way, an upper and lower bound of reasonable performance is clearly visible.

Also, both the external and internal validity of the evaluation would be increased if the toolkit was evaluated in a production environment in a controlled experiment. A deployment to a production environment would uncover potential problems that cannot be seen through simulations. Thus, an actual deployment to real users is a priority in future work. However, the approach with surrogate user models for simulations is sufficient to demonstrate

the toolkit's technical capabilities. The use of the surrogate user models also increases *replicability*, since at least the simulations are fully repeatable (see Appendix).

5 Tooling Support for Bandit Optimization

The open source toolkit Constrained Online Multi-variate Bandit Optimization (COMBO)[1] is a collection of bandit optimization algorithms, constraint solvers, and utilities. This section will first present concepts in the toolkit and then specifics of the included algorithms. COMBO is written in Kotlin and can be used from any JVM language or Node.js. Kotlin is primarily a JDK language but can transcompile to JavaScript and native with LLVM.

To use the toolkit one must first specify a search space of variables and constraints in an embedded domain-specific language (DSL) and map them to software configuration parameters. *Variables* can be of different types: boolean, nominal, finite-domain integers, etc. Though COMBO is optimized for categorical data rather than numeric data. For example, the internal data representation is by default backed by sparse bit fields. *Constraints* can be of types: first-order logic, linear inequalities, etc. For example, if one boolean variable b requires another variable a the constraint $b \Rightarrow a$ can be added.

As mentioned in Section 3, some use cases of the toolkit require context variables for personalization. There is no explicit difference between decision variables and context variables in COMBO—any variable can be set to a fixed value when generating a configuration and have the rest of the variables chosen by the algorithm.

Variables and constraints are always specified within a *model*. There can be any number of nested models in a tree hierarchy. The hierarchy is inspired from the formal variability management language of *feature models* (Cámara and Kobsa 2009). The hierarchy fulfills two additional purposes. First, having a mechanism for the case when a variable has sub-settings but the variable itself is disabled; then the sub-settings should not be updated. It is counted as a missing value by the bandit optimization algorithm. A variable can also be declared as *optional* in which case it has an internal indicator variable that specifies whether it is missing. Second, the hierarchy supports lexical scope to enable composability. That is, models can be built separately in isolation and then joined together in a joint superordinate model without namespace collisions, because they have different variable scopes.

Each model has a proposition that governs if the variables below it are toggled. The constraint $b \Rightarrow a$ can also be expressed implicitly through the tree hierarchy if a is the root variable and b is a child variable, as such:

```
model("a") { bool("b") }
```

The example shows a basic model of a search space with a variable b and the root variable a. It has the implicit constraints $b \Rightarrow a$ and a with two solutions: (a, b) and $(a, \neg b)$. Note that the root variable is always a boolean which is also added as a unit constraint.

Figure 4 shows a more advanced example, there are five variables and three of them are defined in the root model. The variable *Product cards* is the root of a sub model and is the parent to the variables *Two-column* and *Horizontal*, so there are implicit constraints *Two-column* $\Rightarrow$ *Product cards* and *Horizontal* $\Rightarrow$ *Product cards*. Conceptually, in the application that uses COMBO, the variables *Two-column* and *Horizontal* modify the behavior of the *Product cards* so the first two variables can only be true when *Product cards* are enabled.

Further details of the DSL illustrated by the example in Fig. 4 are summarized below:

[1]The source code is available at https://github.com/rasros/combo.

```
model("Auto complete") {
    val width = nominal("Width", "S", "M", "L")
    optionalNominal("Search suggestions", 1, 3, 5, 10)
    val cards = optionalNominal("Product cards", 1, 3, 5)
    model(cards) {
        bool("Two-column")
        val hz = bool("Horizontal")
        impose { width["S"] equivalent !cards }
        impose { width["L"] equivalent (hz and !cards[1]) }
    }
    impose { "Search suggestions" or "Product cards" }
}
```

Fig. 4 Partial model specification for the auto complete search widget (see Section 6.1). The example illustrates: the hierarchy with a child model, different types of variables: *bool* or *nominal*, optional variables, and imposed logic constraints

- Non-root models do not need to declare new variables, they can use any proposition (constraint or variable) as its toggle.
- Explicit constraints are added using an *impose* block. They can be first-order logic, linear inequalities, cardinality constraints, and reified constraints.
- In constraints, variables are referred to by their name as a string within their lexical scope or through an ordinary object reference.
- The indicator variable of optional variables and the specific values of a variable can be used in constraints, as is seen in the usages of the variable *Product cards*.

5.1 Constraint Solvers

The model specifies a constraint satisfaction problem which is solved with, e.g., finite-domain combinatorial constraint programming solvers (Rossi et al. 2006) or boolean satisfiability (SAT) solvers (Biere et al. 2009), depending on what type of constraints and variables are used. The primary use of the constraint solvers is to perform the arg max calculation of multi-variate Thompson sampling, as part of the Optimizer Policy box in Fig. 1.

We see three more uses for constraint solvers for experimentation that are enabled by the toolkit. First, a constraint solver can be used to formally verify the model, which is the main point of Cámara and Kobsa (2009). For example, by verifying that each variable can be both enabled and disabled. Second, randomly ordered permutations of configurations can be used to sample the search space for integration testing purposes. Third, in a large scale MVT with a fractional factorial design (see Section 2.1) and constraints between variables, being able to generate random solutions is required. The problem of generating random solutions uniformly is known as a witness (Chakraborty et al. 2013).

The toolkit includes the SAT solver Sat4j (Le Berre and Parrain 2010) and constraint programming solver Jacop (Kuchcinski 2003). It also features two search-based optimization algorithms: genetic algorithms and local search with tabu search, annealing random walk, and unit constraint propagation (Jussien and Lhomme 2002). The extensions to local search are crucial because there are mutual exclusivity constraints created through nominal variables that would be otherwise hard to optimize with.

When used in combination with the bandit optimization algorithms listed below the solvers should optimize some function rather than just deciding satisfiability. The specifics of how this is done depends on the bandit algorithm. In general, both the black-box search-based methods and the constraint programming solvers can be applied to any function, while

the SAT solvers can be applied to optimize linear functions with integer weights if they support MAX-SAT.

5.2 Bandit Optimization Algorithms

The sections below give some intuition behind how the multi-variate bandit optimization algorithms included in the toolkit work. They include decision tree bandit (Claeys et al. 2017; Elmachtoub et al. 2017), random forest bandit (Féraud et al. 2016), generalized linear model bandit (Chapelle and Li 2011; Hill et al. 2017; Li et al. 2010), and neural network bandit (Riquelme et al. 2018). All of them have custom implementations in COMBO, except the neural network bandit which is implemented using the Deeplearning4j framework.

5.2.1 Decision Tree and Random Forest Bandit

Decision trees recursively partition the search space into homogeneous regions. It has been formulated as a contextual MAB algorithm with Thompson sampling (Elmachtoub et al. 2017) and used for A/B testing in practice (Claeys et al. 2017). The algorithm has a tree for each arm that partition the contextual variables. The trees are updated iteratively using the VFDT (Domingos and Hulten 2000) procedure where each tree initializes with a root node and adds splits greedily. When selecting an arm the statistics of the leaf node corresponding to the user context is used by an optimizer policy.

When adapting this to multi-variate bandit optimization we use only one tree that splits both decision variables and context variables. Due to the partitioning of the search space the posterior distributions used for Thompson sampling can be defined separately.

The policy of selecting a software configuration from a given tree with Thompson sampling can proceed as follows in three steps. First, sample a value from the posterior distribution of all leaf nodes and select the leaf node with the maximum sampled value. Second, calculate the configuration from the leaf node by following the leaf node to the root node and set all parameters according to the splits taken in the path. Third, any x-value unset in the selected leaf node can be chosen at random. For example, consider the middle tree in Fig. 5 with root split on x_3, and then further split on x_1 for $x_3 = 1$, with $\boldsymbol{x} \in \{0, 1\}^3$. If the leaf node where $x_3 = 1$ and either $x_1 = 0$ or $x_1 = 1$ is selected, then x_2 can be randomly selected. For the leaf node where $x_3 = 0$, both x_1 and x_2 is randomly selected while satisfying the constraints.

Random forests are an ensemble learning version of multiple decision trees that often perform better than an individual tree. Each tree sees a random subset of variables and data

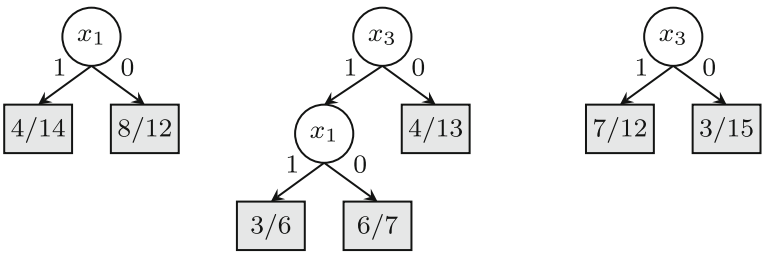


Fig. 5 Example of random forest bandit with three trees. The square leaf nodes show the bootstrapped statistics for binary success/failure. Statistics for potential splits are also kept at each leaf node but not shown here

points. Random forest also has a contextual MAB formulation (Féraud et al. 2016), although the derivation of this work to multi-variate MAB is unclear. Instead, we did a monte carlo tree search (Browne et al. 2012) over time by augmenting each decision tree in the ensemble with statistics at each split node that aggregates the data of all children below it.

When selecting a configuration from the random forest the following procedure is applied iteratively. Aggregate all split nodes at the top level of each decision tree by their decision variable. Sample a value from each decision's pooled statistics and select the best split. Then update the top level node of each tree by following along the split decision. Continue until all trees are exhausted.

Consider the example random forest in Fig. 5. The posterior distributions for the top level decisions are: $x_1 = \{0, 1\}$ with Beta(8, 12) versus Beta(4, 14) and $x_3 = \{0, 1\}$ with Beta(3.5, 14) versus Beta(8, 12.5). Note here that the statistics for the middle tree's x_1 node is not counted in the x_1 decision since that node is for x_1 conditioned on $x_3 = 1$, i.e., $P(\theta_1 | \boldsymbol{y}_1, x_3 = 1)$. Then, four samples are drawn from the distributions and the decision corresponding to the highest sample is selected. Suppose $x_3 = 1$ is selected, then the left tree is unchanged, the middle tree will have a new top node x_1 and the right tree is exhausted. The next decision is only on x_1 with distributions Beta(3.5, 10) versus Beta(7, 9.5), after that decision all trees are exhausted.

5.2.2 Generalized Linear Model and Neural Network Bandit

Bandit optimization with linear models for contextual MAB have seen lots of use for recommender systems (Chapelle and Li 2011; Li et al. 2010) and have been adapted to multi-variate bandit optimization at Amazon (Hill et al. 2017) with Thompson sampling. Any type of regression that fits in the generalized linear model (GLM) framework can be used, such as logistic regression for binary rewards or linear regression for normal distributed rewards.

Each variable in the search space and context space has an estimated weight $\hat{\theta}$. The predicted value is a linear combination of the weights $\hat{\boldsymbol{\theta}}$ and the concatenation of variables $\boldsymbol{x}_t$ and context $\boldsymbol{c}_t$. For measuring the uncertainty of a prediction we also need a continuously updated error variance-covariance matrix Σ of the weights. For ordinary linear regression models, Σ is $\sigma^2(X^T X)^{-1}$, where σ is the standard error and X is a matrix where each row is a user's configuration. As more data points are added the values in the covariance matrix will shrink. For generalized linear models, the Laplace approximation (Chapelle and Li 2011; Russo et al. 2017) yields a similar calculation.

The linear model is updated continuously with second-order stochastic gradient descent using the covariance matrix updates as such:[2]

$$\Sigma_t^{-1} = \Sigma_{t-1}^{-1} + \nabla^2 g_t(\hat{\boldsymbol{\theta}}_{t-1}),$$

$$\hat{\boldsymbol{\theta}}_t = \hat{\boldsymbol{\theta}}_{t-1} - \Sigma_t \nabla g_t(\hat{\boldsymbol{\theta}}_{t-1}),$$

where ∇g is the gradient of the prediction error of a specific input vector (representing a software configuration) at time step t.

With this setup Thompson sampling can be used as a policy as follows. Generate a sampled weight vector $\tilde{\boldsymbol{\theta}}_t$ from the multi-variate normal distribution $\mathcal{N}(\hat{\boldsymbol{\theta}}_t, \Sigma_t)$, where $\hat{\boldsymbol{\theta}}_t$ are the

[2]Russo et al. (2017) is a tutorial on how to make the calculations online efficiently with quadratic complexity to the number of variables. A simplified diagonalized covariance is often used instead for its linear complexity and ability to exploit sparseness (Chapelle and Li 2011; Hill et al. 2017; Li et al. 2010).

model weights and Σ_t is the error variance covariance matrix of the weights. Subsequently, the action $\boldsymbol{x}_t$ chosen at step t is the one which maximizes the expected rewards from the sampled weights using the linear prediction, i.e., $\boldsymbol{x}_t = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \tilde{\boldsymbol{\theta}}^\top (\boldsymbol{x}, \boldsymbol{c}_t)$, subject to the constraints.

This objective function can be solved efficiently with many approaches. Including search-based methods which Hill et al. (2017) used, constraint-programming, or either of MAX-SAT and mixed integer linear programming solvers if the constraints are limited to first-order logic and linear inequalities respectively.

Neural networks has been applied to bandit optimization as well. Riquelme et al. (2018) did a simulation evaluation of various deep learning and linear bandits for contextual MAB. The linear model bandit mentioned previously and a version called *neural linear* were the winners of most experiments. In neural linear, a neural network is fed into a linear bandit where the uncertainty is estimated in the linear model and the network is used for improved representation of the structure.

Having a neural network be continuously updated can be done in many different ways. In COMBO, the linear model in neural linear is updated continuously on every step but the network is updated in batches. The batch update is kept for some configurable time and used to train the neural network in multiple epochs, after which it is discarded.

The objective function is significantly harder for the neural networks than for GLM. Constraint-programming is possible but it will be very slow due to poor constraint propagation through the network. The only appealing option in the toolkit is the search-based methods.

6 Validation

The COMBO toolkit was validated by implementing two feature cases and then evaluated by simulating their usage as realistically as possible using data from the validation company. One of the cases modifies an existing feature to make it data-driven and the other improves and generalizes an existing algorithm. As detailed in the Section 4.2.3, the procedure was done in four steps as follows. First, the variability of the feature case was analyzed. Second, the variability was implemented in the toolkit as a proof-of-concept validation. Third, a data set was collected with configurations and reward measurement pairs and a surrogate user model was trained using the data set. Finally, the surrogate user model was used to repeatedly simulate users in a controlled environment. The simulation results are summarized jointly in Section 6.3 and replication instructions are available in Appendix.

6.1 Feature Case 1: Auto Complete Widget for Product Search

The validation company has recently expanded to provide graphical interfaces to their algorithms, targeted at fashion web shops. One of the algorithms with a new graphical interface that they provide is the auto complete search that pops down when a user starts typing a search query. The case was chosen in a workshop session for two reasons. First, there is no industry consensus on how an auto complete widget is supposed to look. There is not even an agreement on what type of items the widget should show: suggested search terms, brands, product categories, and/or products. It also might be the case that different web shops require different configurations depending on available product data and how big and diverse the product catalogue is. Second, it is an isolated component with low risk that does not affect the rest of the site appearance.

6.1.1 Model Variability

There were three inputs to the decisions taken for modeling the auto complete feature. First, a brainstorm session with a user experience designer was conducted. During the session, several existing auto complete widgets were investigated. The optimization target was decided to be usability, such as whether or not the suggestions saw use or not. It was decided to include only user experience variables and exclude visual design parameters from the model. This was because the validation company preferred if the visual design was consistent with the web shops' general style. Examples of user experience variables are: the types of data to use, whether to have two columns, whether prices and/or sales indicator should be shown with images, etc. Example design parameters are: colors of highlight and borders, font selection and size, etc.

Second, we used guidelines produced from a company specialized in user experience (UX) research for e-commerce: Baymard Institute. They have conducted user tracking sessions and synthesized advice for auto complete and more. The guidelines are not free and the specifics cannot be disclosed, other than that as a requirement from the validation company the guidelines should be adhered to in the designs, either as a decision variable in the optimization or as a constraint. It included things like the maximum number of items to display, how to keep the different data types separate, and some common specific pitfalls.

Third, screenshots from 37 fashion web sites were analyzed to serve as test cases such that the model captures all the salient variability on the sites. Only variables with five or more occurrences were included in the model.

6.1.2 Model Implementation

The resulting model of the auto complete in the COMBO DSL can be seen in a much simplified form in Fig. 4. Some visualizations renditions of the model are given in Fig. 6. The full model has 32 variables (or 53 binary variables), with 28 explicit constraints and 43 implicit constraints from hierarchy or variable specification. Some relevant

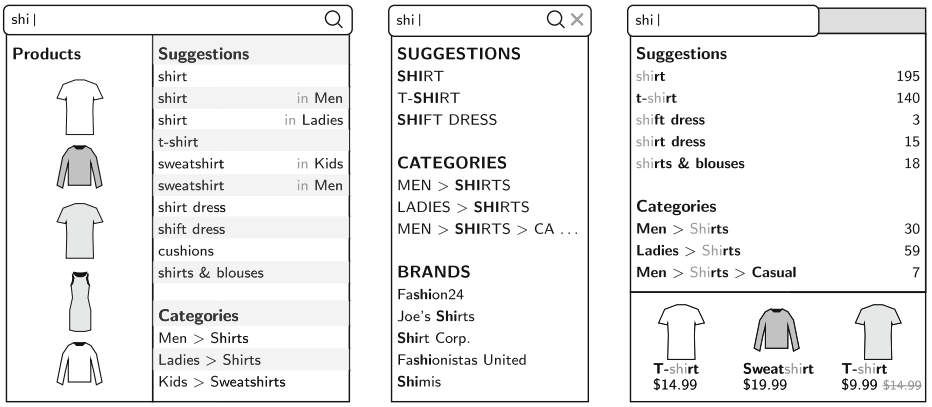


Fig. 6 Visualization of different auto complete widgets. To the left are two randomly generated designs and to the right is one with a high score. The scores of the surrogate user model are in increasing order from left to right, with scores: −0.02, 0.01, and 0.31, where higher scores indicate better perceived user experience. All were generated with constrained equal height for illustration purposes

parameters not visible in Fig. 6 are: underlined highlight, strict search matching, images with fashion models, etc. Among others, there are constraints to calculate the total discretized width and height of the widget (which can be used to generate widgets of specific dimensions), constraints for exclusive variables (e.g., in Fig. 6 the inlined categories that say *in Men* to the left and the counts to the right occupy the same space), and that there must be at least either search term suggestions, product cards, or category suggestions.

After the data was collected it took 10 h to construct the model by the first author. Having test cases available made the process of eliminating invalid variants much easier, such that some valid combinations were not accidentally removed. Also the ability to do formal validation was useful, by querying the system for whether a specific combination was valid or not.

6.1.3 Surrogate User Model Implementation

Since the functionality of the auto complete widget was new at the time, there was no data to base the surrogate user model on for the simulations. Instead a questionnaire on the 37 collected sites were constructed; for each web shop a search on the ambiguous search term 'shi' was conducted. Then participants were asked to score the usability of 20 randomly chosen web shops on a 1–10 scale. In total 16 participants completed the questionnaire which resulted in 320 data points.

A dataset was then constructed by describing each web site in terms of the model. The score was z-normalized to mean 0 and variance 1, as such, the left most widget in Fig. 6 is below average while the middle is slightly above. The dataset was used to train the surrogate user model. Linear regression was chosen since there was not enough data points for anything more sophisticated. Generating a simulated reward for bandit feedback was then done by making a prediction with the surrogate user model and adding noise estimated from the standard error of the fitted model.

Variables were added to handle the following confounding factors: persons scoring the web site, whether suggested terms and images were relevant to search, whether there were duplicated suggestions, what genders the search results were targeted to (male, female, mixed, and unisex). In addition, some pairwise interaction terms were added. These extra variables were not part of the search space but they improved the predictive power of the surrogate user model. The final surrogate user model size was 141 binary variables and 141 constraints (coincidentally equal).

6.2 Feature Case 2: Top-*k* Categories

Many web shops have organized their product catalogue as a category tree (c.f., Fig. 6 under the Categories headings). The validation company provides many algorithms related to the category tree; one example is displaying a subset of the top-k most relevant categories. Where k is the number of categories to display in a given listing. A naïve algorithm would be to simply display the most clicked categories, this might then not sufficiently cover all users' interests. They have more advanced versions in-place already in Hammar et al. (2013). The purpose with this feature case is to show that a generic implementation of the feature is possible using the toolkit. Another goal was to evaluate the algorithms with a more challenging optimization problem than the previous feature case. The search space is much larger and since the data is from real usage the signal-to-noise ratio is lower.

6.2.1 Model Variability

Since this feature is a re-implementation of an existing feature we could select a real web shop and use their category tree. The chosen web shop had sufficient data volumes and a category tree of medium size. The category tree has 938 nodes, with a maximum depth of 4. The web shop that the data came from operates in three different countries. The country that a user comes from was added as a nominal variable to the model. In the simulations the country variable was used for personalization and was generated randomly.

6.2.2 Model Implementation

The model corresponds directly to the category tree, see a simplification in Fig. 7, the actual model is programmatically built different for each specific web shop. A constraint is added to the bottom of each sub model to enforce that one of the sub models' variables are active. In addition to the constraints from the hierarchy, there is also a numeric variable k that control how many categories there can be. A *cardinality* constraint ensures that the number of categories are less than or equal to k. The k-variable can be set for each user or chosen by the decision algorithm. The total model size is 1093 variables with 1101 binary values and 1245 constraints.

6.2.3 Surrogate User Model Implementation

The data set for the surrogate user model was collected with 2737568 configurations and reward pairs. Each data point was constructed by observing what categories were clicked on for each consumer and a reward of 1 was received if the consumer converted their session to a purchase and 0 otherwise. Only data points with at least one click on a categories listing were eligible. In this case, the parameter k is derived from historic user data in the training set for the surrogate user model. This means that the data set was collected for a slightly different scenario than what it is used for. One artifact of this is that the learned surrogate user model is maximized by having $k = 100$, since that correlates with users that spend more time on the site and are likely to convert to customers. However, this effect would not be present in the actual use case. To counteract this, the k parameter is fixed to a specific value ($k = 5$) during simulation.

```
model("Category tree") {
    model("Ladies") {
        bool("Top level")
        model("Jeans") {
            bool("Top level")
            bool("Skinny")
            bool("Loose")
            impose { "Jeans" equivalent or(scope.variables) }
        }
        model("Shirts") { /* More categories follows */ }
        impose { "Ladies" equivalent or(scope.variables) }
    }
    val k = int("Top-k", min = 1, max = 100)
    impose { cardinality(k, LE, leafCategories()) }
}
```

Fig. 7 Partial model specification in COMBO for the top-k categories feature. The function *leafCategories* has its implementaiton omitted, it should return each variable that does not have any sub variables

Since there were lots of data and the point was to make a challenging problem, a neural network was chosen to be the surrogate user model. A standard feed-forward neural network was trained with PyTorch for 30 min with desktop hardware. There were first a dropout layer; then two hidden layers with 15 nodes each, ReLU activation, and batch normalization; and finally a softmax output.

6.3 Simulation Evaluation Results

All bandit optimization algorithms from Section 5.2 and two baselines were evaluated on both surrogate user models, see an overview of the results in Table 1. Again, these results do not provide strong evidence in favor of one algorithm or another, but they show the feasibility of the approach and highlight important choices in algorithm design. Local search was used as optimizer by all the algorithms with the same configuration, except neural linear which was tuned to improve the speed at the expense of performance. The simulations were done on the JVM with desktop hardware with an 8-core computer.

All simulations in the table and figures were repeated for multiple *repetition runs*: 1 000 runs for auto complete and 200 runs for top-k categories. In each repetition, each algorithm start out from a blank slate with no learned behaviour. The algorithm iteratively chooses a configuration and updates the underlying machine learning model at each time step, for time horizon $T = 10\,000$ steps. As such, all numbers in Table 1 are means of means.

Each algorithms have several *hyper-parameters* which are algorithmic parameters that are tweaked before the simulation begins. They were specified with a meta-model in COMBO and optimized with a random forest bandit. The following summary contain the most important hyper-parameters of the respective algorithms:

Random Baseline for comparison. The configurations are generated uniformly at random with the only criteria that they should satisfy the constraints.

Oracle Baseline for comparison. This uses the local search optimizer to maximize the surrogate user model directly, which the other bandits do not have direct access to. The global maximum for auto complete is 0.3953 and for top-k categories it is 0.1031. The numbers for Oracle in Table 1 show how far from optimal the local search optimizer is on

Table 1 Simulation results of bandit optimization algorithms evaluated on both feature cases: auto complete search (AC) and top-k categories for $k = 5$

Algorithm	Choose (ms)		Update (ms)		Mean rewards (SD)	
	AC	Top-k	AC	Top-k	AC	Top-k
Random	0.10	0.81	–	–	0.0136 (9e–4)	0.0879 (1e–5)
Oracle	3.91	81.86	–	–	0.3658 (9e–4)	0.0970 (9e–5)
DT	0.16	2.10	0.03	0.03	0.2131 (0.05)	0.0888 (4e–4)
RF	0.99	43.58	0.13	2.61	**0.2501** (0.04)	**0.0895** (8e–4)
GLM_{diag}	1.52	25.12	0.01	0.01	0.1635 (0.05)	0.0891 (2e–5)
GLM_{full}	1.64	25.35	0.10	8.39	0.2335 (0.04)	0.0891 (2e–5)
NL	8.08	32.68	7.41	12.54	0.0435 (0.06)	0.0881 (6e–4)

Mean rewards is the maximization target. *Choose* and *update* is how long time the algorithm takes to choose a configuration and update it respectively

average. Since the surrogate user model has additional interaction terms the optimization problem is harder for Oracle than for some algorithms.

DT Decision tree bandit. The hyper-parameters were related to how significant a split should be (δ and τ-parameter of VFDT (Domingos and Hulten 2000)).

RF Random forest bandit with 200 decision tree bandits (as in the DT above). The number of trees in the forest does have an impact on performance but has diminishing payoff after 100. The hyper-parameters were both the parameters from the decision tree bandit and parameters for the bootstrapping procedure of standard random forest, i.e., how many variables and data points each tree should have.

GLM$_{diag}$ Generalized linear model bandit with a simplified diagonalized covariance vector. The hyper-parameters were the prior on the weights, regularization factor, and an exploration factor.

GLM$_{full}$ The same as the above GLM$_{diag}$ but with a full covariance matrix. The hyper-parameters were the also same.

NL neural linear bandit with logit output, ReLU activation on the hidden layers, and weight-decay regularization. The neural network optimizer was RMSProp running on a CPU. Since neural linear combines the representation power of a neural network with a GLM$_{full}$ bandit, it inherits the hyper-parameters of the GLM bandit. It also has hyper-parameters in the number of hidden layers and their widths, mini-batch size of updates to the network, learning rate of the optimizer, number of epochs to repeat data points, and initialization noise of the weights.

Table 1 summarizes the results of all algorithms for the feature cases, with both mean calculation time and mean rewards. The dimensions are as follows. *Mean rewards* is the maximization target and is the average expected reward per repetition. The numbers in parentheses are standard deviations between repetition runs for mean rewards. The mean rewards with bold emphasis are statistically significant with p-value below 0.0001. Algorithms with high standard deviation tend to get stuck in local optima for some runs of the simulations. *Choose* measures how long the bandit algorithm takes to generate a software configuration in milliseconds and *update* time how long it takes to update the bandit optimization algorithm. The choose time is critical to keep low in order to not degrade user experience, the update times are not as critical as long as they are not too excessive since they cannot be fully parallelized.

The Figs. 8 and 9 further illustrate the differences in performance of the algorithms. Figure 8 shows the performance over each time step averaged over the simulation repetitions. The figure illustrates that the specific choice of time horizon to $T = 10\,000$ does have an impact on performance, if the simulation would continue for 10 or 100 times as long

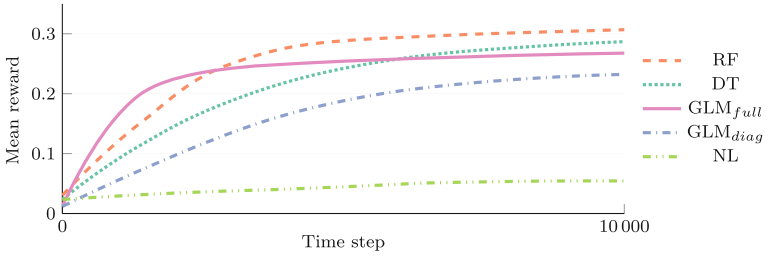
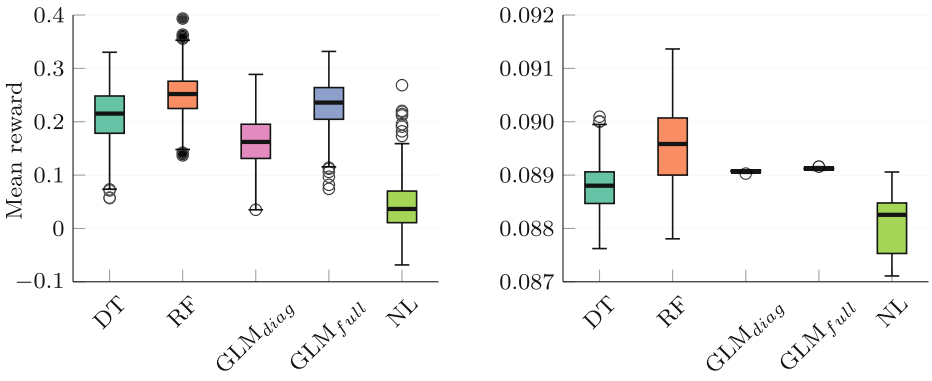


Fig. 8 Mean rewards per repetition over time on the auto complete feature for the bandit optimization algorithms. The maximization target is the area under the curve



(a) Auto complete feature. (b) Top-k categories feature.

Fig. 9 Mean rewards obtained per repetition in the simulations for each bandit optimization algorithms

the ranking of the algorithms might very well have been different, i.e. it might be the case that the neural linear (NL) bandit eventually improves. Figure 9 show the overall variation in mean rewards between repetitions with boxplots on the quartiles and outliers outside the interquartile range.

The results show that the random forest (RF) bandit perform well in terms of rewards for both auto complete and top-k and the neural linear (NL) bandit has poor performance. Clearly, the NL bandit needs more data to converge. As evident from Fig. 9, the performance difference between the algorithms is more pronounced in the top-*k* categories feature. Here it is clear that the random forest bandit is needed for its improved representational power in comparison to the simpler models. However, as can be seen in Fig. 8, the generalized linear model bandit with full covariance matrix (GLM$_{full}$) converges quicker in the auto complete feature.

It should be noted that some algorithms are favored over others in the simulation due to the choices in the surrogate user model. That is, the machine learning model in the top-*k* categories' surrogate user model is the same as the one used by the neural linear (NL) bandit. Also, we did not add any interaction terms to the linear bandits, which could have been done. We reasoned that there will almost always be unmodeled interaction terms in real world applications. Since the surrogate user model for auto complete had pairwise interaction terms only, the linear models would have no interaction terms. In the top-*k* categories feature the performance would probably improve as well with interaction terms.

Initially, before adding hyper-parameter search, the GLM and NL bandits performed much worse than their final performance. Thus, they derived higher benefit from tuned hyper-parameters. As such, the applicability for GLM and NL are dependent on having a simulation environment in which to search for hyper-parameters. In addition, NL is very hard to tune correctly in comparison to the other methods, as evident from the wide performance spread and the large number of hyper-parameters. The outcome of network architecture is also hard to predict. Adding more network nodes or layers increases the representational power of the algorithm, but increases convergence time which also affects performance.

Regarding the time estimates in Table 1, all bandit algorithms are usable within reasonable time; with a choose time below 50ms. The clear winner in both choose and update times is the decision tree (DT) bandit algorithm. The effect of scaling to more variables on

choose and update times can also be seen in the table. The choose time for the RF bandit (see Section 5.2.1) scales poorly, though it can be controlled by limiting the number of trees and the number of nodes and variables per tree. The simulations for the neural linear (NL) bandit are slowest since they are dominated by the non-parallelizable updates.

All the bandit optimization algorithms presented here have trade-offs between performance and choose and update time efficiency. For example, the maximium number of nodes in decision tree (DT), the number of trees in random forest (RF), the number of hidden layers in neural linear (NL), or the number of interaction terms in the generalized linear models (GLM). We have not fully explored this trade-off, other than stating the specific choices we made. We suggest that when deciding on an actual algorithm, developers should start with the threshold on choose time that is acceptable and then find the best algorithm that stays below the threshold.

In summary, the random forest (RF) bandit is the clear winner in the simulations and should serve well as a good default choice. It achieves highest performance for both feature cases and had few outliers with runs of poor performance (c.f., Fig. 9).

7 Discussion

We introduced the Constraint Oriented Multi-variate Bandit Optimization (COMBO) toolkit, used for designing software that improves over time. Using the toolkit, each user receives its own configuration and the toolkit optimizes the overall user experience. It contains machine learning algorithms and constraint solvers, that can be applied to optimize a search space that is specified in a domain specific language. We used the toolkit to model the variability of two features relevant to an e-commerce company called Apptus. Thereby we demonstrated that the toolkit can be applied in industry relevant settings. In this section, the implications of when this is put into practice is discussed.

7.1 From Continuous Experimentation to Continuous Optimization

We define *continuous optimization* as a practice that extends continuous experimentation further by having an algorithm optimize software for better product user experience and business value. The practice entails having a decision algorithm jointly co-optimize the variables in a production environment with user data. As with continuous experimentation, the variables in the optimization can be anything, e.g., details in the visual design and layout, user experience flow, or algorithmic parameters on a web server that impact user experience.

We see two main reasons for that continuous optimization will improve products. First, very large search spaces can be explored by an algorithm. This means that the optimization algorithm might find solutions that developers might otherwise overlook. Miikkulainen et al. (2017) mention this as a common occurrence in their commercial tool for visual design. Second, according to Hill et al. (2017), it enables personalization of software to users by having variables that describe users in the optimization (e.g., device type and location). As many parameters as needed can be added to the search space in order to finely segment users so that the algorithm can find different solutions for different users.

According to Fitzgerald and Stol (2017), discontinuous development is more important than continuous, meaning that product innovation is more important than refinement. We believe that the introduction of a toolkit like COMBO to a development process is not a to contradiction to this. Following the reasoning by Miikkulainen et al. (2017), continuous optimization can de-emphasize narrow incrementalism by offloading parts of the decision

making process so that developers can focus on the more important parts. Thus, continuous optimization offers a complementary approach to software design, by loosely specifying implementation details and letting the algorithm decide instead.

Based on the procedure used to build the feature cases in Section 6, we propose a process for how continuous optimization should be conducted in industry with continuous software development. The validation of the process is preliminary. The four steps of the process are: (1) investigate and prioritize the variability of the feature by user experience (UX) research methods, data mining, or prototype experiments, (2) formulate a model of the variables in the optimization search space and add constraints to prune invalid configurations, (3) tweak the algorithmic performance in offline simulations, and finally (4) validate the solution in a controlled experiment with real users. The process can then restart from step 1.

This process is similar to one that is reportedly used for developing recommender systems at Netflix (Amatriain 2013) and Apptus (Ros and Bjarnason 2018). Simulations are also used at both companies to evaluate changes to their recommender system in fast feedback cycles. If the effect of a change is evaluated to be positive in the simulation, then the change is deployed and subjected to a controlled environment (i.e. an A/B test) in a production environment on real users.

7.2 Considerations for What Metric and Changes to Optimize for

Experimentation and optimization is done with respect to a given metric. Though, quantifying business value in metrics is a well documented challenge for many software companies (Fabijan et al. 2018b; Lindgren and Münch 2016; Olsson et al. 2017; Yaman et al. 2017). Having many metrics in an experiment is one coping strategy, in the hope that together they point in the right direction. Hundreds of metrics are reportedly used for a single experiment at Microsoft (Kevic et al. 2017; Machmouchi and Buscher 2016). For optimization, multi-objective optimization (Nardi et al. 2019; Sun et al. 2018) can be applied offline at compile time, where someone can manually make a trade-off between metrics from a Pareto front. However, for the online bandit optimization algorithms, a single metric is required to serve as rewards (though that metric can be a scalar index).

As mentioned in Section 4.1, there are established metrics in e-commerce that measure business value. Optimizing for revenue or profit is possible but only few users convert to paying customers. Updates to the algorithm will also be delayed due to having to wait until a session expires to determine that they did not convert. In addition, e-commerce companies will have different business models that can result in needing other metrics. For example, they might want to push a certain product since they are overstocked or they might want as many consumers as possible to sign up to their loyalty club to have a recurring source of revenue. All of these volatile factors make optimizing for business value challenging.

In e-commerce, it is unlikely that a change in the user interface will instill a purchasing need in consumers. The way that an optimization algorithm can affect business value is rather by removing hurdles in the user experience that would otherwise make a consumer turn to a competitor. For example, a web shop can be judged to be untrustworthy due to the impression it gives from its design, then customers would not want to buy from there. Consequently, it might be better to optimize with user experience metrics, since all users can contribute data to a user experience metric even if they do not convert. Therefore we argue that, if possible, user experience metrics should be the default choice for optimization before business value metrics.

User experience metrics come with their own set of challenges. For instance, optimizing the number of clicks on one area of the user interface will probably lead to a local improvement, but possibly at the expense of other areas of the site. This effect where changes shift clicks around between areas is known as *cannibalization* (Dmitriev and Wu 2016; Kohavi et al. 2014). If the area that gets cannibalized has higher business value than the cannibalizing area the optimization can even be detrimental. This has been researched at Apptus (Brodén et al. 2017) for their recommender system in particular. If the recommender system is optimized for clicks, it can distract consumers with interesting products, rather than products that they might buy.

Ultimately, different parts of the user interface can be optimized for different things. Each interface component is designed to fulfill a goal and that is the goal to be quantified. Returning to the feature cases from Section 6, in the auto complete feature the goal was to aid users in the discovery of products through the search. Thus, whether they used the aid or not—measured in click-through-rate—is a reasonably risk free optimization metric. In the top-*k* categories, using clicks seems riskier since some categories can distract users or might lead them to believe that the store does not sell certain products that they do sell. For that reason, business value metrics seem fitting. Whether this reasoning is correct or not is something that can be validated in an A/B test once the optimization system is put into production.

7.3 Future Directions

Future work is required to strengthen the evaluation of COMBO to make a stronger claim about the applicability and suitability of the toolkit. A more thorough continuous optimization process would be useful as well. Below we present remaining technical barriers for wide-spread adoption of tools like COMBO that require further study, in no particular order.

7.3.1 Ramifications on Software Quality and Testing

The continuous optimization practice that we advocate for is not without risks. Both the need to maintain machine learning models and the increased variability of software, caused by optimization, are challenging to handle on their own. Sculley et al. (2014) from Google describe how maintaining machine learning models is a source of technical debt, this has also been studied extensively by software quality researchers (Masuda et al. 2018). Regarding software testing, the model-based approach (Chen et al. 2009; Kang et al. 2002) to experimentation (Cámara and Kobsa 2009) and optimization can be used both for formal verification and software testing (see Section 5.1). Much has been published on model-based testing (Utting et al. 2012), also for user interfaces specifically (Silva et al. 2008). Since a model is built regardless for optimization, focusing more on integrating the model-based testing techniques in the toolkit would be fruitful.

7.3.2 Concept Drift for Bandit Optimization

In machine learning, concept drift occurs when the environment that the machine learning model has been trained in changes over time, i.e., if users change their behaviour. There are general approaches to detect and be more robust against concept drift, e.g., Minku and Yao (2011), and specific solutions to software engineering related applications (Kanoun and van der Schaar 2015; Lane and Brodley 1998). Those approaches cannot be directly applied to this work since detection of concept drift is not enough. For univariate

multi-armed bandits, the solution is usually to apply a moving window to the arms' descriptive statistics (Brodén et al. 2017; Burtini et al. 2015). This adds more complexity to the solution since another hyper-parameter needs to be estimated, i.e. how long the window should be. Also, unlearning specific data points for multi-variate multi-armed bandits is harder than unlearning for descriptive statistics. Thus, more work is required to study the impact and approaches for concept drift in this domain.

7.3.3 Continuous Model Updates

Adding or removing variables to the optimization search space must be done without restarting the optimization. For linear models, decision trees, and random forests, model updates are straightforward to both implement and make inference about. For neural networks the effect of a change will be unpredictable, the algorithm might not even converge to a new solution so it could be better to restart the model. This is further discussed in the technical debt for machine learning paper (Sculley et al. 2014). Furthermore, users will have their existing configurations be invalidated when the underlying model is updated. Their configuration in the new model's search space should preferably not be drastically different from their old one to avoid user confusion. We plan to add support to the toolkit for generating configurations that minimize the distance between a configuration of an old search space to a new one; while at the same time maximizing the expected reward of the new configuration.

7.3.4 Bandit Optimization Algorithms

Several bandit optimization algorithms are included in the COMBO toolkit. Still, we have only begun to explore the all the design options for algorithms. For example, ensembles of several bandit optimization algorithms could be applied to improve performance by initially using algorithms that learn quickly and then gradually switching to algorithms with better representational power.

7.3.5 Algorithm Tuning and Cold Start

Based on the simulation evaluation in Section 6.3 we advice against just using default settings on a bandit optimization algorithm. The performance can improve drastically by tuning hyper-parameters in simulations, though we refrain from giving numbers on the improvement since the default parameters are somewhat arbitrarily set in the first place. However, this will be hard when implementing a new feature with no data. This was the case for the auto complete feature in Section 6.1 where data was manually collected. This might be too expensive for regular software development. Regardless, the constraint oriented approach in the COMBO toolkit can be used here to exclude incompatible design choices observed during feature development (possibly in conjunction with informative Bayesian priors). The situation is analogous to the *cold start* problem in recommender systems (Schein et al. 2002). That occurs when new products are added that do not have any associated behavioural data. Thus, connections could be drawn to this research field.

8 Conclusion

In e-commerce, continuous experimentation is widespread to optimize web shops; high-profile companies have recently adopted online machine learning based optimization

methods to both increase scale of the optimizations, and to have personalized software to different user's needs. This technology is readily available but can be hard to implement on anything other than superficial details in the user interface. In this work, the open-source toolkit COMBO is introduced to support the algorithmic optimization at a validation company in e-commerce. The toolkit can be used for building data-driven software features that learn from user behaviour in terms of user experience or business value. We have shown that modeling software hierarchies in a formal model enables algorithmic optimization of complex software. Thus, the toolkit extends optimization to more use cases. There are still many further opportunities for future work in this domain to enable adoption in other fields than e-commerce. However, we insist that the toolkit can be used today to improve the user experience and business value of software.

Acknowledgements Thanks to Per Runeson, Elizabeth Bjarnason, Luigi Nardi, and the three anonymous reviewers for providing useful feedback to this manuscript.

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

Funding Information Open access funding provided by Lund University

Appendix: Replication of Simulations

To replicate the simulations in Section 6.3, run the following commands on a Unix terminal with a JDK and git:

```
$ git clone https://github.com/rasros/combo.git
$ cd combo
$ ./gradlew assemble
$ # runs the random forest bandit on the auto complete dataset:
$ ./jvm-demo/simulation.sh RF AC
$ # view a list of options, algorithms, and datasets:
$ ./jvm-demo/simulation.sh -h
```

References

Amatriain X (2013) Beyond data: from user information to business value through personalized recommendations and consumer science. In: Proceedings of the 22nd ACM international conference on conference on information and knowledge management—CIKM'13. ACM Press, hhtps://doi.org/10.1145/2505515.2514701

Arcuri A, Fraser G (2011) On parameter tuning in search based software engineering. In: International symposium on search based software engineering. Springer, pp 33–47, hhtps://doi.org/10.1007/978-3-642-23716-4_6

Auer F, Felderer M (2018) Current state of research on continuous experimentation: a systematic mapping study. In: 2018 44th Euromicro conference on software engineering and advanced applications (SEAA). IEEE, pp 335–344, https://doi.org/10.1109/SEAA.2018.00062

Bakshy E, Eckles D, Bernstein MS (2014) Designing and deploying online field experiments. In: Proceedings of the 23rd ACM conference on the World Wide Web. ACM

Biere A, Heule M, van Maaren H (2009) Handbook of satisfiability, vol 185. IOS Press, Amsterdam

Borg M (2016) Tuner: a framework for tuning software engineering tools with hands-on instructions in r. J Softw: Evol Process 28(6):427–459. https://doi.org/10.1002/smr.1784

Bosch-Sijtsema P, Bosch J (2015) User involvement throughout the innovation process in high-tech industries. J Prod Innov Manag 32(5):793–807. https://doi.org/10.1111/jpim.12233

Brodén B, Hammar M, Nilsson BJ, Paraschakis D (2017) Bandit algorithms for e-Commerce recommender systems. In: Proceedings of the 11th ACM conference on recommender systems, pp 349–349, https://doi.org/10.1145/3109859.3109930

Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of monte carlo tree search methods. IEEE Trans Comput Intell AI Games 4(1):1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

Burtini G, Loeppky J, Lawrence R (2015) A survey of online experiment design with the stochastic multi-armed bandit. ArXiv e-prints. arXiv:1510.00757v4 [stat.ML]

Cámara J, Kobsa A (2009) Facilitating controlled tests of website design changes: A systematic approach. In: Lecture notes in computer science. Springer, Berlin, pp 370–378, https://doi.org/10.1007/978-3-642-02818-2_30

Chakraborty S, Meel KS, Vardi MY (2013) A scalable and nearly uniform generator of sat witnesses. In: International conference on computer aided verification. Springer, Berlin, pp 608–623

Chamberlain S, Sharp H, Maiden N (2006) Towards a framework for integrating agile development and user-centred design. In: International conference on extreme programming and agile processes in software engineering. Springer, Berlin, pp 143–153, https://doi.org/10.1007/11774129_15

Chapelle O, Li L (2011) An empirical evaluation of thompson sampling. In: Proceedings of the 24th international conference on neural information processing systems, NIPS'11, pp 2249–2257

Chen L, Ali Babar M, Ali N (2009) Variability management in software product lines: a systematic review. In: Proceedings of the 13th international software product line conference. Carnegie Mellon University, pp 81–90

Chen W, Wang Y, Yuan Y (2013) Combinatorial multi-armed bandit: general framework and applications. In: International conference on machine learning, pp 151–159

Claeys E, Gançarski P, Maumy-Bertrand M, Wassner H (2017) Regression tree for bandits models in A/B testing. In: International symposium on intelligent data analysis. Springer, Berlin, pp 52–62

Dmitriev P, Wu X (2016) Measuring metrics. In: Proceedings of the 25th ACM international on conference on information and knowledge management, pp 429–437, https://doi.org/10.1145/2983323.2983356

Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 71–80

Elmachtoub AN, McNellis R, Oh S, Petrik M (2017) A practical method for solving contextual bandit problems using decision trees. arXiv preprint arXiv:1706.04687

Fabijan A, Dmitriev P, Olsson HH, Bosch J (2018a) Effective online controlled experiment analysis at large scale. In: 2018 44th Euromicro conference on software engineering and advanced applications (SEAA). IEEE, https://doi.org/10.1109/seaa.2018.00020

Fabijan A, Dmitriev P, Olsson HH, Bosch J (2018b) Online controlled experimentation at scale: an empirical survey on the current state of A/B testing. In: 2018 44th Euromicro conference on software engineering and advanced applications (SEAA). IEEE, https://doi.org/10.1109/seaa.2018.00021

Fagerholm F, Guinea AS, Mäenpää H, Münch J (2017) The RIGHT model for continuous experimentation. J Syst Softw 123:292–305. https://doi.org/10.1016/j.jss.2016.03.034

Feitelson DG, Frachtenberg E, Beck KL (2013) Development and deployment at Facebook. IEEE Internet Comput 17(4):8–17

Felfernig A, Mandl M, Tiihonen J, Schubert M, Leitner G (2010) Personalized user interfaces for product configuration. In: Proceedings of the 15th international conference on intelligent user interfaces, pp 317–320, https://doi.org/10.1145/1719970.1720020

Féraud R, Allesiardo R, Urvoy T, Clérot F (2016) Random forest for the contextual bandit problem. In: Artificial intelligence and statistics, pp 93–101

Fitzgerald B, Stol KJ (2017) Continuous software engineering: a roadmap and agenda. J Syst Softw 123: 176–189. https://doi.org/10.1016/j.jss.2015.06.063

Forrester A, Sobester A, Keane A (2008) Engineering design via surrogate modelling: a practical guide. Wiley, New York

Gerostathopoulos I, Uysal AN, Prehofer C, Bures T (2018) A tool for online experiment-driven adaptation. In: 2018 IEEE 3rd international workshops on foundations and applications of self* systems (FAS* W). IEEE, pp 100–105, https://doi.org/10.1109/FAS-W.2018.00032

Gupta S, Ulanova L, Bhardwaj S, Dmitriev P, Raff P, Fabijan A (2018) The anatomy of a large-scale experimentation platform. In: 2018 IEEE international conference on software architecture (ICSA). IEEE, https://doi.org/10.1109/icsa.2018.00009

Hadar I, Hasson T, Ayalon O, Toch E, Birnhack M, Sherman S, Balissa A (2018) Privacy by designers: software developers' privacy mindset. Empir Softw Eng 23(1):259–289. https://doi.org/10.1007/s10664-017-9517-1

Hammar M, Karlsson R, Nilsson BJ (2013) Using maximum coverage to optimize recommendation systems in e-commerce. In: Proceedings of the 7th ACM conference on recommender systems. ACM, pp 265–272

Hill DN, Nassif H, Liu Y, Iyer A, Vishwanathan S (2017) An efficient bandit algorithm for realtime multivariate optimization. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining—KDD '17. ACM Press, https://doi.org/10.1145/3097983.3098184

Hoos HH (2012) Programming by optimization. Commun ACM 55(2):70–80. https://doi.org/10.1145/2076450.2076469

Hutter F, Xu L, Hoos HH, Leyton-Brown K (2014) Algorithm runtime prediction: methods & evaluation. Artif Intell 206:79–111. https://doi.org/10.1016/j.artint.2013.10.003

Iitsuka S, Matsuo Y (2015) Website optimization problem and its solutions. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 447–456, https://doi.org/10.1145/2783258.2783351

Jiménez M, Rivera LF, Villegas NM, Tamura G, Müller HA, Bencomo N (2019) An architectural framework for quality-driven adaptive continuous experimentation. In: 2019 IEEE/ACM joint 4th international workshop on rapid continuous software engineering and 1st international workshop on data-driven decisions, experimentation and evolution (RCoSE/DDrEE). IEEE, pp 20–23, https://doi.org/10.1109/RCoSE/DDrEE.2019.00012

Jussien N, Lhomme O (2002) Local search with constraint propagation and conflict-based heuristics. Artif Intell 139(1):21–45

Kang KC, Lee J, Donohoe P (2002) Feature-oriented product line engineering. IEEE Softw 19(4):58–65

Kanoun K, van der Schaar M (2015) Big-data streaming applications scheduling with online learning and concept drift detection. In: 2015 Design, automation & test in Europe conference & exhibition (DATE). IEEE, pp 1547–1550, https://doi.org/10.7873/DATE.2015.0786

Kephart JO, Chess DM (2003) The vision of autonomic computing. Computer 36(1):41–50. https://doi.org/10.1109/MC.2003.1160055

Kevic K, Murphy B, Williams L, Beckmann J (2017) Characterizing experimentation in continuous deployment: a case study on bing. In: Proceedings of the 39th international conference on software engineering: software engineering in practice track. IEEE Press, pp 123–132

Kharitonov E, Macdonald C, Serdyukov P, Ounis I (2015) Optimised scheduling of online experiments. In: Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval - SIGIR'15. ACM Press, https://doi.org/10.1145/2766462.2767706

Kohavi R, Longbotham R, Sommerfield D, Henne RM (2008) Controlled experiments on the web: survey and practical guide. Data Min Knowl Discov 18(1):140–181. https://doi.org/10.1007/s10618-008-0114-1

Kohavi R, Deng A, Frasca B, Longbotham R, Walker T, Xu Y (2012) Trustworthy online controlled experiments: Five puzzling outcomes explained. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 786–794, https://doi.org/10.1145/2339530.2339653

Kohavi R, Deng A, Longbotham R, Xu Y (2014) Seven rules of thumb for web site experimenters. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining, pp 1857–1866, https://doi.org/10.1145/2623330.2623341

Kuchcinski K (2003) Constraints-driven scheduling and resource assignment. ACM Trans Des Autom Electron Syst (TODAES) 8(3):355–383

Lane T, Brodley CE (1998) Approaches to online learning and concept drift for user identification in computer security. In: KDD'98: proceedings of the fourth international conference on knowledge discovery and data mining, pp 259–263

Le Berre D, Parrain A (2010) The sat4j library, release 2.2, system description. J Satisf Boolean Model Comput 7:59–64

Li L, Chu W, Langford J, Schapire RE (2010) A contextual-bandit approach to personalized news article recommendation. In: Proceedings of the 19th international conference on World wide web. ACM, pp 661–670

Lindgren E, Münch J (2016) Raising the odds of success: the current state of experimentation in product development. Inf Softw Technol 77:80–91. https://doi.org/10.1016/j.infsof.2016.04.008

Machmouchi W, Buscher G (2016) Principles for the design of online A/B metrics. In: Proceedings of the 39th international ACM SIGIR conference on research and development in information retrieval - SIGIR'16. ACM Press, https://doi.org/10.1145/2911451.2926731

Masuda S, Ono K, Yasue T, Hosokawa N (2018) A survey of software quality for machine learning applications. In: 2018 IEEE international conference on software testing, verification and validation workshops (ICSTW). IEEE, pp 279–284, https://doi.org/10.1109/ICSTW.2018.00061

Mattos DI, Bosch J, Olsson HH (2017) Your system gets better every day you use it: towards automated continuous experimentation. In: 2017 43rd Euromicro conference on software engineering and advanced applications (SEAA). IEEE, pp 256–265, https://doi.org/10.1109/SEAA.2017.15

Mattos DI, Bosch J, Olsson HH (2019) Multi-armed bandits in the wild: pitfalls and strategies in online experiments. Inf Softw Technol 113:68–81. https://doi.org/10.1016/j.infsof.2019.05.004

Miikkulainen R, Lamba G, Iscoe N, Shagrin A, Cordell R, Nazari S, Schoolland C, Brundage M, Epstein J, Dean R (2017) Conversion rate optimization through evolutionary computation. In: Proceedings of the genetic and evolutionary computation conference on - GECCO '17. ACM Press, https://doi.org/10.1145/3071178.3071312

Miikkulainen R, Iscoe N, Shagrin A, Rapp R, Nazari S, McGrath P, Schoolland C, Achkar E, Brundage M, Miller J et al (2018) Sentient ascend: Ai-based massively multivariate conversion rate optimization. In: Thirty-second AAAI conference on artificial intelligence

Minku LL, Yao X (2011) Ddd: a new ensemble approach for dealing with concept drift. IEEE Trans Knowl Data Eng 24(4):619–633. https://doi.org/10.1109/TKDE.2011.58

Nair V, Yu Z, Menzies T, Siegmund N, Apel S (2018) Finding faster configurations using flash. IEEE Trans Softw Eng. https://doi.org/10.1109/TSE.2018.2870895

Nardi L, Koeplinger D, Olukotun K (2019) Practical design space exploration. In: 2019 IEEE 27th international symposium on modeling, analysis, and simulation of computer and telecommunication systems (MASCOTS). IEEE, pp 347–358, https://doi.org/10.1109/MASCOTS.2019.00045

Olsson HH, Bosch J, Fabijan A (2017) Experimentation that matters: a multi-case study on the challenges with A/B testing. In: Lecture notes in business information processing. Springer International Publishing, pp 179–185, https://doi.org/10.1007/978-3-319-69191-6_12

Rahman MT, Querel LP, Rigby PC, Adams B (2016) Feature toggles: practitioner practices and a case study. In: Proceedings of the 13th international conference on mining software repositories, pp 201–211, https://doi.org/10.1145/2901739.2901745

Révész Á, Pataki N (2017) Containerized A/B testing. In: 6th Workshop on software quality analysis, monitoring, improvement, and applications, SQAMIA 2017. CEUR-WS, p 14

Ries E (2011) The lean startup: how today's entrepreneurs use continuous innovation to create radically successful businesses, 1st edn. Crown Business

Riquelme C, Tucker G, Snoek J (2018) Deep bayesian bandits showdown: an empirical comparison of bayesian deep networks for thompson sampling. arXiv preprint arXiv:1802.09127

Ros R, Bjarnason E (2018) Continuous experimentation scenarios: a case study in e-Commerce. In: 2018 44th Euromicro conference on software engineering and advanced applications (SEAA). IEEE, https://doi.org/10.1109/seaa.2018.00064

Ros R, Runeson P (2018) Continuous experimentation and A/B testing: a mapping study. In: Proceedings of the 4th international workshop on rapid continuous software engineering (RCoSE). ACM, pp 35–41, https://doi.org/10.1145/3194760.3194766

Ros R, Bjarnason E, Runeson P (2017) Automated controlled experimentation on software by evolutionary bandit optimization. In: International symposium on search based software engineering. Springer, pp 190–196, https://doi.org/10.1007/978-3-319-66299-2_18

Rossi F, Van Beek P, Walsh T (2006) Handbook of constraint programming. Elsevier, New York

Runeson P, Engström E, Storey MA (2020) The design science paradigm as a frame for empirical software engineering. In: Felderer M, Travassos GH (eds) Contemporary empirical methods in software engineering, chap 5, Nature. In press

Russo D, Van Roy B, Kazerouni A, Osband I (2017) A tutorial on thompson sampling. arXiv preprint arXiv:1707.02038

Schein AI, Popescul A, Ungar LH, Pennock DM (2002) Methods and metrics for cold-start recommendations. In: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, pp 253–260, https://doi.org/10.1145/564376.564421

Schermann G, Leitner P (2018) Search-based scheduling of experiments in continuous deployment. In: 2018 IEEE international conference on software maintenance and evolution (ICSME). IEEE, https://doi.org/10.1109/icsme.2018.00059

Schwabe D, Guimarães RM, Rossi G (2002) Cohesive design of personalized web applications. IEEE Internet Comput 6(2):34–43. https://doi.org/10.1109/4236.991441

Sculley D, Holt G, Golovin D, Davydov E, Phillips T, Ebner D, Chaudhary V, Young M (2014) Machine learning: the high interest credit card of technical debt. In: SE4ML: software engineering for machine learning (NIPS 2014 Workshop)

Silva JL, Campos JC, Paiva AC (2008) Model-based user interface testing with spec explorer and concurtasktrees. Electron Notes Theor Comput Sci 208:77–93. https://doi.org/10.1016/j.entcs.2008.03.108

Singer J, Sim SE, Lethbridge TC (2008) Software engineering data collection for field studies. In: Guide to advanced empirical software engineering. Springer, pp 9–34

Snoek J, Larochelle H, Adams RP (2012) Practical bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, pp 2951–2959

Sun J, Zhang H, Zhou A, Zhang Q, Zhang K, Tu Z, Ye K (2018) Learning from a stream of nonstationary and dependent data in multiobjective evolutionary optimization. IEEE Trans Evol Comput 23(4):541–555. https://doi.org/10.1109/TEVC.2018.2865495

Sutton RS, Barto AG et al (1998) Introduction to reinforcement learning, 2 edn. MIT Press, Cambridge

Tamburrelli G, Margara A (2014) Towards automated A/B testing. In: Proceedings of the 6th international symposium on search-based software engineering (SSBSE), pp 184–198, https://doi.org/10.1007/978-3-319-09940-8_13

Tang D, Agarwal A, O'Brien D, Meyer M (2010) Overlapping experiment infrastructure: more, better, faster experimentation. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining - KDD '10. ACM Press, https://doi.org/10.1145/1835804.1835810

Utting M, Pretschner A, Legeard B (2012) A taxonomy of model-based testing approaches. Softw Test Verif Reliab 22(5):297–312. https://doi.org/10.1002/stvr.456

Wieringa RJ (2014) Design science methodology for information systems and software engineering. Springer, Berlin. https://doi.org/10.1007/978-3-662-43839-8

Williams A (2009) User-centered design, activity-centered design, and goal-directed design: a review of three methods for designing web applications. In: Proceedings of the 27th ACM international conference on design of communication, pp 1–8, https://doi.org/10.1145/1621995.1621997

Yaman S, Sauvola T, Riungu-Kalliosaari L, Hokkanen L, Kuvaja P, Oivo M, Männistö T (2016) Customer involvement in continuous deployment: A systematic literature review. In: Proceedings of the 22nd international conference on requirements engineering: foundation for software quality (REFSQ), pp 249–265, https://doi.org/10.1007/978-3-319-30282-9_18

Yaman SG, Munezero M, Münch J, Fagerholm F, Syd O, Aaltola M, Palmu C, Männistö T (2017) Introducing continuous experimentation in large software-intensive product and service organisations. J Syst Softw 133:195–211. https://doi.org/10.1016/j.jss.2017.07.009

Affiliations

Rasmus Ros[1] · **Mikael Hammar**[2]

Mikael Hammar
mikael.hammar@apptus.se

[1] Lund University, Lund, Sweden

[2] Apptus Technologies AB, Lund, Sweden