Targeting Optimization for Internet Advertising by Learning from Logged Bandit Feedback

Margherita Gasparini, Alessandro Nuara, Francesco Trovò, Nicola Gatti, Marcello Restelli
Dipartimento di Elettronica, Informatica e Bioingegneria
Politecnico di Milano, Milano, Italy
Email: margherita.gasparini@mail.polimi.it,
{alessandro.nuara, francesco1.trovo, nicola.gatti, marcello.restelli}@polimi.it

***Abstract*—In the last two decades, online advertising has become the most effective way to sponsor a product or an event. The success of this advertising format is mainly due to the capability of the Internet channels to reach a broad audience and to target different groups of users with specific sponsored announces. This is of paramount importance for media agencies, companies whose primary goal is to design ad campaigns that target only those users who are interested in the sponsored product, thus avoiding unnecessary costs due to the display of ads to uninterested users. In the present work, we develop an automatic method to find the best user targets (a.k.a. *contexts*) that a media agency can use in a given Internet advertising campaign. More specifically, we formulate the problem of target optimization as a Learning from Logged Bandit Feedback (LLBF) problem, and we propose the TargOpt algorithm, which uses a tree expansion of the target space to learn the partition that efficiently maximizes the campaign revenue. Furthermore, since the problem of finding the optimal target is intrinsically exponential in the number of the features, we propose a tree-search method, called A-TargOpt, and two heuristics to drive the tree expansion, aiming at providing an anytime solution. Finally, we present empirical evidence, on both synthetically generated and real-world data, that our algorithms provide a practical solution to find effective targets for Internet advertising.**

I. Introduction

In the last two decades, the use of the Internet has increased exponentially, and online advertising has become the most effective way to sponsor a product or an event. In 2016, companies invested more than 70 billion USD only in the US for Internet advertising [1], with an increase of the revenue of more than 20% with respect to the previous year. The success of this advertising format is mainly due to the capability of the Internet channels to reach a broad audience in a fast and efficient way and, at the same time, to target different groups of users with specific sponsored announces. In this field, the main players are the *advertisers*, who have some product/event to advertise, the *media agencies*, whose task is to manage advertisement campaigns for the advertisers, and the Web *publishers*, whose role is to provide slots in Web pages reserved for advertising purposes. Commonly, ad slots are allocated to the advertisers through auctions (i.e., Vickrey-Clarke-Groves or Generalized Second Price [2]), in which for each ad it is required to specify a *bid*—how much the advertiser is willing to pay for a click on the ad—, and a *budget* per day that the advertiser is willing to spend. In addition to the *search engines* (e.g., Google, Bing, Yahoo!), also the *social medias* (e.g., Facebook, LinkedIn) allow the display of content interleaved with ads, and the annual spent over 2016 for social advertising has been more than 9 billion USD (only in US). A key to success of Internet advertising is the possibility of targeting very accurately the ads to the users, thanks to a huge amount of data on the users' behavior available to the advertisement platforms [3]. However, such amount of data makes the problem of finding the best targeting unaffordable for humans, thus needing automatic methods. To this purpose, in the present paper, we develop an automatic method to find the best user targets (a.k.a. *contexts*) that a media agency can use in a given Internet advertising campaign.

An advertising campaign is structured in a number of *subcampaigns*, each specifying an ad, a target, and a bid and a daily budget. The media agencies are required to design advertising campaigns from scratch, creating the ads (text and banners), partitioning the user target into homogeneous groups, and finally addressing the problem of the bid/budget optimization for every subcampaign, whose nature is combinatorial as showed in [4]. While the creation of the ads is usually left to marketing experts, we focus on the joint task of selecting a suitable target for an advertising campaign and, at the same time, to optimize the bid/budget of each subcampaign. Such a problem is addressed in literature as Learning from Logged Bandit Feedback (LLBF). Nonetheless, our problem presents two peculiarities: first, the data about users, which are available to the media agencies, are aggregated (i.e., the behavior of a single user cannot be perfectly tracked); second, the problem of jointly optimizing bid and budget in a campaign is combinatorial. The task of selecting the proper target for an advertising campaign has been explored in the past, e.g., by [5]–[8], but these approaches require a full track of the users' behavior during her Internet navigation. Furthermore, we cannot resort to the works on LLBF [9], [10] previously available in the literature since they do not apply to combinatorial optimization problems such, instead, our problem is.

Original Contributions. In the present work, we formulate the problem of target optimization as an LLBF problem, and we propose the TargOpt algorithm, which uses a tree expansion of the target space to learn the partition providing the maximum number of conversions efficiently. In doing so, we use a risk adverse approach. Furthermore, since the problem of finding the optimal target is intrinsically expo-

nential in the number of the features, any algorithm may, in principle, require exponential time. To cope with this issue, we propose a tree-search method, called A-TargOpt and two heuristics to drive the tree expansion, aiming at providing an anytime solution. Finally, we provide empirical evidence, on both synthetically generated and real-world data, that our algorithms provide an effective solution to find effective targets for Internet advertising.

II. Related Works

There exists a wide literature investigating the segmentation and selection of users based on their online shopping behaviour [5], [6], both on social networks [7] and on search engines [8]. In [5], the authors aim at identifying online consumers who are most likely to purchase a specific product for the first time in the near future after seeing an advertisement. The proposed techniques are based on a multistage transfer learning system. In [6], the authors analyse the effect of display advertising on customer conversions, considering individuals who visited a certain website in the past, individuals who have never visited the site but were targeted, and individuals who have never visited the site and were not targeted any ads about the website. In [7], the authors propose a method to build an affinity network among users on social networks who accessed the same content, which should also have the same purchasing interests. Therefore, once a user in the network is interested in a product, the ad is targeted to the other components of her affinity network. In [8], the authors show that the click-through-rate of an ad can be consistently improved using segmentation of users based on Behavioral Targeting, a technique which uses information collected about an individual web-browsing behaviour to select which ads fit best that user. Nonetheless, all these works assume a constant tracking of the online activities of the user, in such a way that when an advertiser decides the target of an ad, she can address specific users rather than categories of users. Conversely, in the problem we study, we cannot target one particular consumer, not having this kind of punctual information.

The other main line of research about Internet advertising is focused on the analysis [11] or the design [12]–[16] of new algorithms for the bid and/or budget optimization given a fixed contextual partition of the target users. All the aforementioned works are designed to operate in an offline fashion. Only recently, in [4], the AdComb algorithm has been proposed for the online joint bid/budget optimization of pay-per-click multi-channel advertising campaigns. The authors formulate the optimization problem as a combinatorial bandit problem, in which they use Gaussian Processes to estimate stochastic functions, Bayesian bandit techniques to address the exploration/exploitation problem, and a dynamic programming technique to solve a variation of the Multiple-Choice Knapsack (MCK) problem. All the aforementioned methods are able to work only if the target partition is given as input.

The more general problem of using logged bandit feedback to provide a policy behaving differently over different contexts to use in the future runs is tackled by [9], [10]. In [9], the authors design POEM, an algorithm using counterfactual regret minimization to estimate a linear model between features and available actions. In [10], the RADT algorithm is proposed to learn a decision tree which maximizes the specifically crafted lower confidence bounds on the profit. Both these algorithms assume the feedbacks to be generated by a classical MAB algorithm and are not able to tackle the complexity of the combinatorial nature of the choice we have in the problem we study. Therefore, the algorithms previously available in the literature cannot be directly applied to the bandit feedbacks provided by a generic Internet advertising campaign.

III. Problem Formulation

An Internet advertising campaign $\mathcal{C} := \{s_1, \dots, s_N\}$ is described by a set of N subcampaigns s_i, each of which is identified by a tuple of K features $s_i := (z_{i1}, \dots, z_{iK})$, e.g., specifying the gender, age or the interests of the users we target by the subcampaign s_i. Each feature $z_{ij} \subseteq Z_j$ is a nonempty set of values characterising the subcampaign, where Z_j is the set of the feasible values for the j-th feature. For instance, if the j-th feature corresponds to the gender, with values M for male and F for female, we have $Z_j = \{M, F\}$ as the set of feasible values, and, thus, the corresponding feature can be $z_{ij} = \{M\}$ if the subcampaign s_i targets only male users, $z_{ij} = \{F\}$ if it targets only the female ones, and $z_{ij} = \{M, F\}$ if it targets both. We assume that the subcampaigns are targeting different sets of users. This implies that, for each pair of subcampaigns in $\mathcal{C}$, these are disjoint, formally:

Definition 1: Two subcampaigns s_i and s_k are *disjoint* ($s_i \cap s_k = \emptyset$) if it exists an index $j \in \{1, \dots, K\}$ s.t. $z_{ij} \cap z_{kj} = \emptyset$.

To optimally set the target of the advertising campaign $\mathcal{C}$, we are provided with a set of logged bandit feedbacks generated by the application of an unknown decision policy $\mathfrak{U}_0$ over a time horizon of length T. At a generic round $t \in \{1, \dots, T\}$, the policy $\mathfrak{U}_0$ selects a bid/budget pair for each subcampaign $s_i \in \mathcal{C}$.[1] Consider the following definitions:

Definition 2: A subcampaign s_i is called *atomic* if $|z_{ij}| = 1$ for each $j \in \{1, \dots K\}$, i.e., in which each feature has a single element.

Definition 3: Given two subcampaigns s_i and s_k we say that s_i is included in or equal to s_k ($s_i \subseteq s_k$) if:

$$z_{ij} \subseteq z_{kj} \quad \forall j \in \{1, \dots, K\}.$$

We assume to have, at each round t during which the policy $\mathfrak{U}_0$ runs, the following information on every atomic subcampaign s_i such that there is $s_j \in \mathcal{C}$ with $s_i \subseteq s_j$:

- $b_t(s_i)$ is the bid which has been selected;
- $p_t(s_i)$ is the amount of budget spent;
- $n_t(s_i)$ is the number of clicks obtained;

[1] We do not make any assumption on the policy $\mathfrak{U}_0$, except that it should provide feedback about all the possible bid/budget pairs and all the subcampaigns $s_i \in \mathcal{C}$ we want to analyse for target optimization. For instance, a policy $\mathfrak{U}_0$ which never allocates budget on a specific subcampaign $s_{\hat{i}}$ over the whole time horizon does not allow the optimization of the target for $s_{\hat{i}}$.

- $v_t(s_i)$ is a cumulative revenue obtained by the conversions.

Finally, there exists a function $n(s_i, x, y)$ returning the average number of clicks for a generic subcampaign s_i obtained when setting bid x and budget y and a parameter $v(s_i)$ denoting the average value per click of subcampaign s_i.

The problem of jointly optimizing the values of the bid and daily budget for each subcampaign of a given advertising campaign $\mathcal{C}$ has already been addressed in [4], where the authors cast such a problem as a MCK problem [17]. More formally, the problem aims at finding a bid/budget pair $(x(s_i), y(s_i))$ for each subcampaign s_i such that:

$$\begin{aligned} J^*(\mathcal{C}) = & \max_{\{(x(s_i),y(s_i))\}_{i=1}^{N}} J(\mathcal{C}) \\ & \text{s.t. } \sum_{i=1}^{N} y(s_i) \leq \overline{B} \\ & \underline{x}(s_i) \leq x(s_i) \leq \overline{x}(s_i) \quad \forall i \in \{1, \dots N\} \\ & \underline{y}(s_i) \leq y(s_i) \leq \overline{y}(s_i) \quad \forall i \in \{1, \dots N\} \end{aligned}$$

where

$$J(\mathcal{C}) := \sum_{i=1}^{N} v(s_i)\, n(s_i, x(s_i), y(s_i))$$

is the revenue generated by the advertising campaign in a single day, $\overline{B}$ is the cumulative daily budget spent for all the subcampaigns in a day, $\underline{x}(s_i)$ and $\overline{x}(s_i)$ are the minimum and maximum bid values available for the subcampaign s_i, $\underline{y}(s_i)$ and $\overline{y}(s_i)$ are the minimum and maximum daily budget values that can be allocated to the subcampaign s_i.

The optimization problem described above cannot control the structure of the campaign $\mathcal{C}$. This means that campaign $\mathcal{C}$ keeps to be unchanged during the whole time horizon. Conversely, in the present work, we face the problem of changing the configuration of the subcampaigns to maximize the expected objective function $J^*(\mathcal{C})$. The space $\mathcal{M}$ in which we search for the optimal configuration of subcampaigns is:

$$\mathcal{M} = \{\mathcal{C} = \{s_1, \dots, s_N\} \text{ s.t. } \forall i, j, i \neq j \quad s_i \cup s_j = \emptyset\},$$

i.e., the space of all the advertising campaigns whose subcampaigns are disjoint. The optimization problem we study is:

$$\mathcal{C}^* := \arg\max_{\mathcal{C} \in \mathcal{M}} J^*(\mathcal{C}).$$

To fulfill this goal we will make use of the information over the time horizon T provided by the policy $\mathfrak{U}_0$.

IV. Proposed Method

In this section, we describe an algorithm that, given a set of bids and budgets and a cumulative budget $\overline{B}$, finds an advertising campaign $\mathcal{C}^*$ which maximizes the expected revenue. This will be done by an exploration of the space $\mathcal{M}$ over a specifically designed tree and the use of a novel algorithm, called Target Optimization (TargOpt) able to work on it. In the case the space of all the possible feasible solutions is extremely large (due to a high number of features), a complete exploration of the space $\mathcal{M}$ is generally unfeasible. Therefore, we also provide a tree search algorithm, called Any-time Target Optimization (A-TargOpt), and effective heuristics able to efficiently visit the space $\mathcal{M}$.

A. Number of Click Function Approximation

At first, we discretize of the bid/budget space. For the sake of concision and without loss of generality, we adopt the same discretization grid over the bid/budget space for all the atomic subcampaigns s. Specifically, we have $\underline{x}(s) = \underline{x} > 0$, $\overline{x}(s) = \overline{x}$, $\underline{y}(s) = \underline{y} > 0$ and $\overline{y}(s) = \overline{B}$, and we use a uniform grid over the bid/budget space $X \times Y$ as follows:

$$X = \left\{ \underline{x} + \frac{h}{N_x}(\overline{x} - \underline{x}) \right\}_{h=0}^{N_x},$$

$$Y = \left\{ \underline{y} + \frac{h}{N_y}(\overline{y} - \underline{y}) \right\}_{h=0}^{N_y},$$

where $N_x \in \mathbb{N}^+$ and $N_y \in \mathbb{N}^+$ determine the granularity of the discretization for the bid and budget, respectively. Furthermore, we use lower bounds on the number of clicks in place of the empirical average value. Indeed, the maximization of the empirical average is not a suitable criterion to design a policy [18], since it might be arbitrarily far from the actual expected value. Instead, lower bounds take into account the uncertainty that affects the actual value, providing a risk-averse policies that minimize the probability of realizing a very low revenue or even a loss.

We compute the lower bounds $\underline{n}(s_i, x, y)$ and $\underline{v}(s_i)$ for the number of clicks $n(s_i, x, y)$ and the click values $v_i(s_i)$, respectively, with a given confidence δ, for each bid/budget pair in $X \times Y$ as follows. Let us focus on the number of clicks. Given a subcampaign s_i, we compute a function $\underline{n}(s_i, x, y, \delta)$ that, for each element $(x, y) \in X \times Y$, returns a lower bound holding with probability δ on the number of clicks $n(s_i, x, y)$. This task is solved by the algorithm proposed in [4], where the estimation of the number of clicks is performed by means of Gaussian Processes [19]. More specifically, the number of clicks $n(s_i, x, y)$ corresponding to a specific bid/budget pair $(x, y) \in X \times Y$ is modeled as a Gaussian distribution, whose parameters, mean $\mu(s_i, x, y)$ and variance $\sigma^2(s_i, x, y)$, are estimated relying on historical observations $(b_t(s_i), p_t(s_i), n_t(s_i))_{t=1}^{T}$.[2] The lower bound on the number of clicks is computed as:

$$\underline{n}(s_i, x, y) := \hat{\mu}(s_i, x, y) - z_\delta \hat{\sigma}(s_i, x, y),$$

where $\hat{\mu}(s_i, x, y)$ is the estimates for the mean, $\hat{\sigma}(s_i, x, y)$ is the estimates of the standard deviation and z_δ the quantile of order δ of the standard Gaussian distribution. The same methodology can be applied to estimate a lower bound $\underline{v}(s_i)$ on the value $v(s_i)$ of subcampaign s_i resorting to the sequence of samples $(v_t(c_i))_{t=1}^{T}$. See [4] for details. We underline that the procedure to obtain lower bounds we describe above can be substituted by any other suitable procedure. For instance, one can adopt a procedure using only historical data on a specific bid/budget pair (x, y) to estimate the lower bound $\underline{n}(s_i, x, y)$ employing, e.g., the Hoeffding bound [20].

[2] In this section, we summarize the procedure to estimate the number of clicks, and we refer to [4] for technical details.

Moreover, we remove the dependency of our optimization problem on the bid x by finding the best bid for every campaign s_i and every value of the daily budget y. We denote by $\underline{n}(s_i, y)$ the lower bound on the number of clicks of campaign s_i when the daily budget is y and the best bid is used, formally:

$$\underline{n}(s_i, y) := \max_{x \in X} \underline{n}(s_i, x, y).$$

In the next section, we denote the function returning the (lower bounds of the) revenue generated by a subcampaign s_i with:

$$P(s_i, y) := \underline{v}(s_i)\, \underline{n}(s_i, y). \quad (1)$$

B. Tree Construction and Optimization

Let us define two operators working with campaigns and subcampaigns used in what follows.

Definition 4: Given a campaign $\mathcal{C}_i \in \mathcal{M}$ and a subcampaign s_j we say that $\mathcal{C}_i$ is included in s_j ($\mathcal{C}_i \subseteq s_j$) if:

$$\forall s_k \in \mathcal{C}_i \quad s_k \subseteq s_j.$$

Definition 5: Given a subcampaign s_j and a feature index $f \in \{1, \ldots, K\}$, the partition operator $D := d(s_j, f)$ returns the set D of all the possible campaigns that can be generated by partitioning the sub-campaign s_j w.r.t. the feature x_{jf}. Formally, $D := \{\mathcal{C}_1, \ldots, \mathcal{C}_{part(x_{jf})}\}$ s.t. each $\mathcal{C}_i \subseteq s_j$ and $\forall s_k \in \mathcal{C}_i\ x_{ih} = x_{jh}, \forall h \neq f$, where $part(x_{jf})$ is the number of partitions of the set x_{jf}.

For instance, given a subcampaign $s_j = \{\{M, F\}, \{Y, A\}\}$—where the features are gender (M for male, F for female) and age (Y for young, A for adult)—and index $f = 1$, the partition operator $d(s_j, f)$ returns $D = \{\{s_1, s_2\}, \{s_3\}\}$, where $s_1 = (\{M\}, \{Y, A\})$, $s_2 = (\{F\}, \{Y, A\})$, and $s_3 = (\{M, F\}, \{Y, A\})$. This means that D is composed of two campaigns, the first composed, in its turn, of two subcampaigns, the second composed of a single subcampaign.

We use the tree $\mathcal{T} := (\mathcal{E}, \mathcal{O})$, composed of two different sets of nodes: an even level nodes set $\mathcal{E} := \{E_i\}$, in which each node E_i corresponds to a different campaign, and an odd level nodes set $\mathcal{O} := \{O_j\}$, in which each node O_j corresponds to a subcampaign which is a child to some even node E_i. Each level of the tree corresponds to a single feature. The even levels nodes $E_i := (\mathcal{C}_i, Child_i, f_i, \boldsymbol{J}_i, Arg_i)$ and odd levels nodes $O_j := (s_j, Child_j, f_j, \boldsymbol{J}_j, Arg_j)$ are defined as tuples in which:

- $\mathcal{C}_i$ is a campaign;
- s_j is a subcampaign;
- $Child_i$ and $Child_j$ are the sets of the children nodes;
- $f_i, f_j \in \{0, \ldots, K\}$ are feature indexes indicating the level of the node and, at the same time, which feature has been selected;
- $\boldsymbol{J}_i := (\boldsymbol{J}_i(y))_{y \in Y}, \boldsymbol{J}_j = (\boldsymbol{J}_j(y))_{y \in Y}$, is the vector of the lower bound of the revenues for each $y \in Y$ (initially empty, which is used in the optimization procedure);
- $Arg_i, Arg_j \in \mathcal{M}^{N_y}$ is a vector of campaigns (initially empty, which is used in the optimization procedure).

The root of the tree $\mathcal{T}$ is the node $E_0 = (\mathcal{C}, Child_0, 0, \boldsymbol{J}_0, Arg_0)$, where $\mathcal{C} := \{s_1, \ldots, s_N\}$ is the original advertising campaign, $s_j \in \mathcal{C}$ are its subcampaigns, and the set of children nodes $Child_0$ is composed of odd nodes $O_j = (s_j, Child_j, 0, \boldsymbol{J}_j, Arg_j)$. Given a non-atomic subcampaign s_j, the set of the children $Child_j$ of a generic odd node $O_j = (s_j, Child_j, f_j, \boldsymbol{J}_j, Arg_j)$ is composed of the even nodes $E_i = (\mathcal{C}_i, Child_i, f_j + 1, \boldsymbol{J}_i, Arg_i)$ s.t. the campaign $\mathcal{C}_i$ is in $D = d(s_j, f_j)$. Instead, if s_j is atomic, $Child_j$ is empty, meaning that O_j is a leaf. Given a campaign $\mathcal{C}_i$, the set of the children $Child_i$ of a generic even node $E_i = (\mathcal{C}_i, Child_i, f_i, \boldsymbol{J}_i, Arg_i)$ is composed of odd nodes $O_j = (s_j, Child_j, f_i, \boldsymbol{J}_j, Arg_j)$ in which s_j is one of the subcampaigns contained in $\mathcal{C}_i$. The construction of the tree $\mathcal{T}$ consists in the successive expansion of the root node E_0 until no odd node can be expanded further.

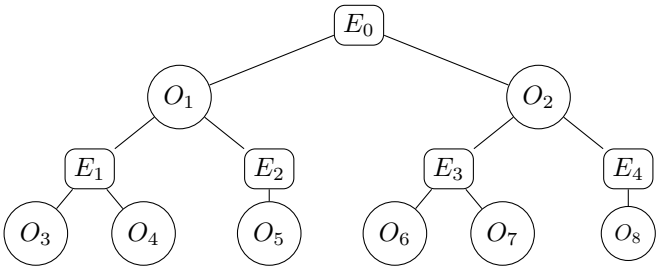

Fig. 1. An example of a tree representing a completely expanded advertising campaign, in which the original campaign $\mathcal{C}$ had two subcampaigns $s_1 = (\{M\}, \{Y, A\})$ and $s_2 = (\{F\}, \{Y, A\})$, where the subcampaigns are defined by gender (M for Male, F for Female) and age (Y for Young, A for Adult).

In Figure 1, we show an example of a completely expanded tree $\mathcal{T}$, generated starting from the campaign $\mathcal{C} = \{s_1, s_2\}$, with $s_1 = (\{M\}, \{Y, A\})$ and $s_2 = (\{F\}, \{Y, A\})$. The root of the tree E_0 (even rectangular node) corresponds to the original campaign $\mathcal{C}$, and its children nodes O_1 and O_2 (circular odd nodes) corresponds to the subcampaigns s_1 and s_2, respectively. The next level contains the nodes representing all the possible campaigns and subcampaigns that can be generated by partitioning the target of the subcampaigns s_1 for O_1 and s_2 for O_2. More specifically, nodes O_3 and O_4 correspond to subcampaign $s_3 = (\{M\}, \{Y\})$ and $s_4 = (\{M\}, \{A\})$, respectively, node E_1 corresponds to campaign $\mathcal{C}_1 = \{s_3, s_4\}$; node O_5 corresponds to subcampaign $s_5 = s_1$ and E_2 corresponds to campaign $\mathcal{C}_2 = \{s_5\}$; nodes O_6 and O_7 correspond to subcampaign $s_6 = (\{F\}, \{Y\})$ and $s_7 = (\{F\}, \{A\})$, respectively and node E_3 corresponds to campaign $\mathcal{C}_3 = \{s_6, s_7\}$; node O_8 corresponds to subcampaign $s_8 = s_2$ and E_4 corresponds to campaign $\mathcal{C}_4 = \{s_8\}$.

We describe TargOpt algorithm to find the campaign maximizing the revenue. The algorithm traverses the tree from the leaves to the root and computes the lower bound of the revenue for each campaign in $\mathcal{T}$. The pseudocode of the TargOpt algorithm is presented in Algorithm 1. It takes as input a tree $\mathcal{T} = (\mathcal{E}, \mathcal{O})$ and it returns the optimal campaign $\mathcal{C}^*$. The algorithm starts from the lowermost level of the tree ($l = K$) and applies the procedure OddNodeUpdate(O_j), detailed in what follows, for all the odd nodes O_j s.t. $f_j = l$ (Line 6). This procedure fills the vector $\boldsymbol{J}_{new}$ with the revenues

Algorithm 1 TargOpt Algorithm

1: **Input:** tree $\mathcal{T} = (\mathcal{E}, \mathcal{O})$
2: **Output:** campaign $\mathcal{C}^*$
3: $l \leftarrow K$
4: **while** $l \geq 0$ **do**
5: **for all** $O_j \in \mathcal{O} \mid f_j = l$ **do**
6: $\boldsymbol{J}_{new}, Arg_{new} \leftarrow$ OddNodeUpdate(O_j)
7: $O_j \leftarrow (s_j, Child_j, f_j, \boldsymbol{J}_{new}, Arg_{new})$
8: **for all** $E_i \in \mathcal{E} \mid f_i = l$ **do**
9: $\boldsymbol{J}'_{new}, Arg'_{new} \leftarrow$ EvenNodeUpdate(E_i)
10: $E_i \leftarrow (\mathcal{C}_i, Child_i, f_i, \boldsymbol{J}'_{new}, Arg'_{new})$
11: $l \leftarrow l - 1$
12: $\mathcal{C}^* \leftarrow Arg_0(\overline{B})$
13: **return** $\mathcal{C}^*$

corresponding to each budget $y \in Y$ and the corresponding campaign Arg_{new}, i.e., a vector containing campaigns which provides the revenues in $\boldsymbol{J}_{new}$. The nodes are then updated to include this new information (Line 7). After that, these results are used to update the even nodes E_i s.t. $f_i = l$ with the subroutine EvenNodeUpdate(E_i) (Line 9), detailed in what follows. The subroutine aggregates the revenues $\boldsymbol{J}_j$ provided by the children $O_j \in Child_i$, executing a variation of the MCK solving algorithm. Indeed, it provides the vector of the lower bounds on the revenue $\boldsymbol{J}'_{new}$ and the vector of the campaigns Arg'_{new} corresponding to those revenues for each budget $y \in Y$. After that, the algorithm updates each analysed even node E_j (Line 10). The procedure moves to the upper level ($l \leftarrow l - 1$) and iterates until it reaches the root ($l = 0$), where it returns the campaign $Arg_0(\overline{B})$, contained in the root node E_0, corresponding to the optimal revenue $\boldsymbol{J}_0(\overline{Y})$ given a total budget of $\overline{B}$.

The pseudocode of the subroutine EvenNodeUpdate(E_i) is presented in Algorithm 2. It is a variation of the dynamic-programming algorithm used to solve the MCK problem [17] specifically crafted for our scenario. Given an even node E_i it computes the vector of the optimal lower bound of the revenue $\boldsymbol{J}_{new}$ corresponding to each budget $y \in Y$ and the corresponding vector of campaigns Arg_{new}. At first, it sets null revenue and empty optimal campaign for all the budgets $y \in Y$ (Lines 3-5). After that, it analyses each odd node O_j in the set of the children $Child_i$ and evaluates for each budget $y \in Y$ whether the lower bound of the revenue $\boldsymbol{J}_j(y)$ provided by the campaign s_j in the child O_j is better than any other one computed so far, or whether there exists an allocation of budgets over more than one subcampaign that can perform better (Line 10). Then, it stores in $Arg_{new}(y)$ the set of subcampaigns which is providing the largest lower bound on the revenue $\boldsymbol{J}_j(y)$ for the budget y (Lines 12-18). Once this process is repeated for all the nodes $O_j \in Child_i$, the subroutine returns the vector of the largest lower bound on the revenue $\boldsymbol{J}_{new}$ and the vector of the corresponding campaigns Arg_{new} (Line 19).

The subroutine OddNodeUpdate is provided in Algorithm 3. It requires as input an odd node O_j. If O_j does not have any child, the subroutine fills each element of the vector of the lower bounds of the revenue $\boldsymbol{J}_{new}(y)$ using the function $P(s_j, y)$ defined in Equation (1) (Lines 3-6). Conversely, if the set of the children $Child_j$ is not empty , the vector $\boldsymbol{J}_{new}$ is computed by choosing for each $y \in Y$ the maximum value of the revenue $\boldsymbol{J}_i(y)$ among the even nodes $E_i \in Child_j$ (Lines 8-11). The subroutine also stores the campaign vector Arg_{new}, whose elements are the one providing the revenues in the vector $\boldsymbol{J}_{new}$. At last, it returns the vector of the lower bound of the revenue $\boldsymbol{J}_{new}$ and the vector of the campaigns Arg_{new} (Line 12).

Algorithm 2 EvenNodeUpdate Subroutine

1: **Input:** even node E_i
2: **Output:** vector of the lower bounds of the revenues $\boldsymbol{J}_{new}$, vector of the campaigns Arg_{new}
3: $\boldsymbol{J}_{new} \leftarrow \boldsymbol{0}$
4: **for** $y \in Y$ **do**
5: $Arg_{new}(y) \leftarrow \emptyset$
6: **for** $O_j \in Child_i$ **do**
7: $\boldsymbol{J}_{old} \leftarrow \boldsymbol{J}_{new}$
8: $Arg_{old} \leftarrow Arg_{new}$
9: **for** $y \in Y$ **do**
10: $y^* = \arg \max_{y' \in Y, y' \leq y} [\boldsymbol{J}_{old}(y') + \boldsymbol{J}_j(y - y')]$
11: $\boldsymbol{J}_{new}(y) = \boldsymbol{J}_{old}(y^*) + \boldsymbol{J}_j(y - y^*)]$
12: **if** $y^* = 0$ **then**
13: $Arg_{new}(y) = Arg_j(y)$
14: **else**
15: **if** $y^* = y$ **then**
16: $Arg_{new}(y) = Arg_{old}(y)$
17: **else**
18: $Arg_{new}(y) \leftarrow Arg_{old}(y) \cup Arg_j(y)$
19: **return** $\boldsymbol{J}_{new}$, Arg_{new}

Algorithm 3 OddNodeUpdate Subroutine

1: **Input:** odd node O_j
2: **Output:** vector of the lower bounds of the revenues $\boldsymbol{J}_{new}$, vector of the campaigns Arg_{new}
3: **if** $Child_j = \emptyset$ **then**
4: **for** $y \in Y$ **do**
5: $\boldsymbol{J}_{new}(y) \leftarrow P(s_j, y)$
6: $Arg_{new}(y) \leftarrow s_j$
7: **else**
8: **for** $y \in Y$ **do**
9: $i^* \leftarrow \arg\max_{i \mid E_i \in Child_j} \boldsymbol{J}_i(y)$
10: $\boldsymbol{J}_{new}(y) \leftarrow \boldsymbol{J}^*_i(y)$
11: $Arg_{new}(y) \leftarrow Arg_{i^*}(y)$
12: **return** $\boldsymbol{J}_{new}$, Arg_{new}

C. Approximated Algorithm for Large Feature Space

In the case the feature space is such that $K \gg 1$, the expansion of the tree $\mathcal{T}$ up to the atomic subcampaigns might

be computationally expensive, since the number of the odd nodes scales as $O(2^{|Z|K})$, where $|Z| := \min_j |Z_j|$ is the minimum cardinality of the subcampaign features. To perform the target optimization also when the execution of the TargOpt algorithm is unfeasible, we design a variation of the TargOpt algorithm that, starting from a tree composed only by the root node E_0, iteratively expands the nodes in a classical tree-search fashion.

Algorithm 4 A-TargOpt Algorithm

1: **Input:** campaign $\mathcal{C}$, maximum number of nodes to expand $N_{\max}$.
2: **Output:** best campaign discovered $\mathcal{C}^*$
3: $\mathcal{O} \leftarrow \{(s_i, \emptyset, 0, \emptyset, \emptyset)\}_{i=1}^{N}$
4: $\mathcal{E} \leftarrow \{(\mathcal{C}, \{O_1, \dots O_N\}, 0, \emptyset, \emptyset)\}$
5: $\mathcal{C}^* \leftarrow$ TargOpt$((\mathcal{E}, \mathcal{O}))$
6: **while** $|\mathcal{O}| < N_{\max}$ **do**
7: $\quad \mathcal{L} \leftarrow \{O_k \in \mathcal{O} \,|\, Child_k = \emptyset\}$
8: $\quad$ Select $O_j \leftarrow H(\mathcal{L})$
9: $\quad D \leftarrow d(s_j, f_j + 1)$
10: $\quad Child_j \leftarrow \emptyset$
11: $\quad$ **for** $\overline{\mathcal{C}} \in D$ **do**
12: $\quad\quad \overline{Child} \leftarrow \emptyset$
13: $\quad\quad$ **for** $\hat{s} \in \overline{\mathcal{C}}$ **do**
14: $\quad\quad\quad O \leftarrow (\hat{s}, \emptyset, f_j + 1, \emptyset, \emptyset)$
15: $\quad\quad\quad \overline{Child} \leftarrow \mathcal{O} \cup O$
16: $\quad\quad \mathcal{O} \leftarrow \mathcal{O} \cup \overline{Child}$
17: $\quad\quad E \leftarrow (\overline{\mathcal{C}}, \overline{Child}, f_j + 1, \emptyset, \emptyset)$
18: $\quad\quad Child_j \leftarrow Child_j \cup E$
19: $\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup E$
20: $\quad \mathcal{O} \leftarrow \mathcal{O} \setminus O_j$
21: $\quad O_j = (s_j, Child_j, f_j, \emptyset, \emptyset)$
22: $\quad \mathcal{O} \leftarrow \mathcal{O} \cup O_j$
23: $\quad \mathcal{C}^* =$ TargOpt$((\mathcal{E}, \mathcal{O}))$
24: **return** $\mathcal{C}^*$

The pseudocode of our algorithm, called A-TargOpt, is presented in Algorithm 4. It takes as input a campaign $\mathcal{C}$ and a maximum number of odd nodes $N_{\max} \in \mathbb{N}$ to expand.[3] At first, the algorithm initializes the tree with an odd node for each subcampaign in $\mathcal{C}$ (Line 3) and a single even node E_0 for the original campaign (Line 4). In the case the expansion is already too computationally expensive for the available budget ($|\mathcal{O}| > N_{\max}$), it executes the TargOpt algorithm on $\mathcal{T}$ to provide a tentative solution $\mathcal{C}^*$ (Line 5). At each iteration, it expands the more promising nodes according to a strategy function $H(\mathcal{L})$, which takes a set of odd leaf nodes $\mathcal{L}$, i.e., the odd nodes $O_k \in \mathcal{O}$ s.t. $Child_k = \emptyset$, and returns a single odd node $O_j \in \mathcal{L}$ which will be expanded. Here, we propose two different strategies $H(\cdot)$ we adopt to select the more promising node to be expanded:

- Breadth-First Search (BFS), which in our specific setting consists in expanding one of the nodes with the largest target, i.e., $O_j \in \mathcal{L}$ s.t. $\nexists\, O_k \in \mathcal{L}$ s.t. $f_k < f_j$;
- Optimistic Search (OS), which expands the nodes with the maximum average revenue or, formally, $O_j \in \mathcal{L}$ s.t. $j = \arg\max_h \frac{\sum_{y \in Y} \boldsymbol{J}_h(y)}{N_y}$.

Once an odd node O_j has been chosen (Line 8), the algorithm uses the partition operator $D = d(s_j, f_j)$ to expand the subcampaign s_j (Line 9). Moreover, it generates and adds in $Child_j$ all the even nodes $E = (\overline{\mathcal{C}}, \overline{Child}, f_j + 1, \emptyset, \emptyset)$ corresponding to campaigns in D (Lines 17-18) and it adds to the child set $\overline{Child}$ each node E all the odd nodes O corresponding to the subcampaign $\hat{s} \in \overline{\mathcal{C}}$ (Lines 13-15). Finally, the algorithm adds to the tree $\mathcal{T}$ the newly generated node and its children (Lines 19-22), and executes the TargOpt algorithm on the updated version of the tree. These operations are repeated until the tree $\mathcal{T}$ has at most $N_{\max}$ odd nodes.

V. Experimental Evaluation

We experimentally evaluate our algorithm on both real-world and synthetic problems. At first, we show the improvement provided by the TargOpt algorithm on a real-world advertising problem and, after that, we evaluate the performance of the A-TargOpt algorithm in a synthetically generated environment.

A. Evaluation in a Real-world Setting

Experimental Setting. We evaluate the TargOpt algorithm on a real-world dataset, provided by the company MediaMatic [21]. The database corresponds to an advertising campaign for a financial product, whose name is omitted for privacy reasons.[4] The original campaign $\mathcal{C}$ is composed of $N = 8$ subcampaigns. The dataset we use for the experiments consists of the data recorded from $01/09/17$ to $08/11/17$, for a total of $T = 69$ days. The (single) feature z_{i1} that we analyze in this experiment is the hour of the day the ad is displayed to users. More specifically, for this feature, we consider a set of $|Z_1| = 8$ values corresponding to 3-hours-long time slots (0.00 a.m.-3.00 a.m., 3.01 a.m.-6.00 a.m., etc.). A finer granularity would make the optimization problem intractable. Since a preliminary analysis suggests that the behaviour of users is different during weekdays and weekends, e.g., the total volumes of the search are significantly different between the twos, we separate the gathered data to apply the TargOpt algorithm independently to the two scenarios. The policy used for the collection of the data is the AdComb algorithm [4], where we set a budget $\overline{B} = 1100$, while we discretize the bid space evenly in $(\underline{x}, \overline{x}) = (0.1, 5)$ with $|X| = 50$ values, and the budget space evenly in $(\underline{y}, \overline{y}) = (2, 1100)$ with $|Y| = 550$ possible budgets. We use the same discretization of the bid/budget space also while executing the TargOpt algorithm. We compare the optimal campaign $\mathcal{C}^*$, resulting

[3] We limit the number of the odd nodes since they are the most computationally expensive, being those that execute the algorithm solving the MCK problem.

[4] To fulfill the NDA we have with the media agency, some of the values reported in what follows of the experiment have been scaled, given that the transformation we applied does not change the conclusion we draw.

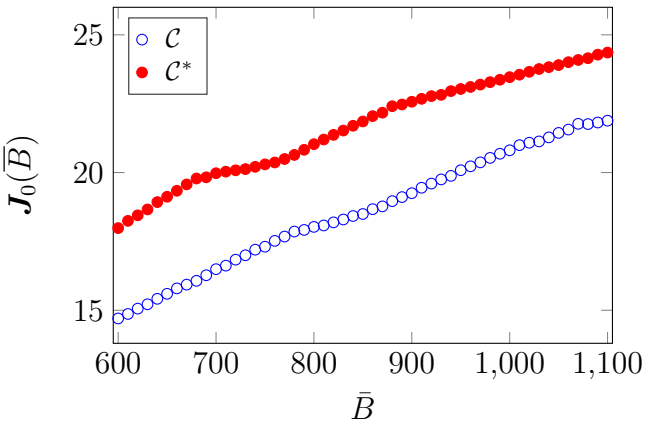

Fig. 2. Revenue of the TargOpt on the real-world dataset for the weekdays.

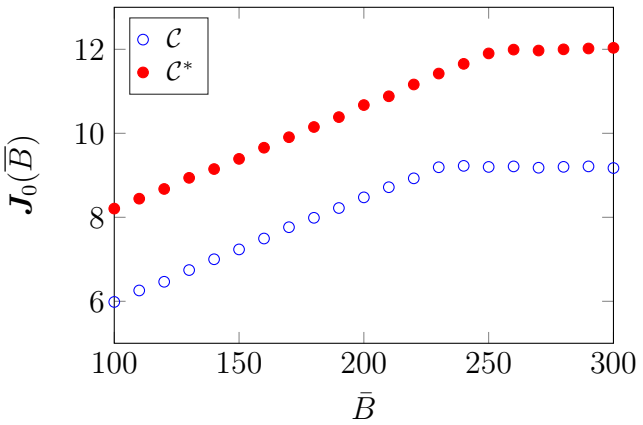

Fig. 3. Revenue of the TargOpt on the real-world dataset for the weekends.

from the execution of the TargOpt algorithm, with the one provided by the original campaign $\mathcal{C}$. The performance index we use to evaluate the performance of the two campaigns is the lower bound on the revenue $\boldsymbol{J}_0(\overline{Y})$, where we assume a unitary value for the conversion.

Results. The results of the experiments are presented in Figure 2 for the weekdays and Figure 3 for the weekends. In both the cases, the TargOpt algorithm provides a significant improvement over the original campaign $\mathcal{C}^*$, i.e., about 13% more conversions during the weekdays and about 30% during the weekend. This improvement does not reduce as the budget invested per day $\overline{B}$ increases. In the weekend scenario, the revenue for campaign $\mathcal{C}$ is almost constant for $\overline{B} \geq 230$ ($\boldsymbol{J}_0(\overline{B}) \approx 9$). This phenomenon is due to the fact that, as we increase the total daily budget, part of what is spent is targeting subcampaigns in which no conversion occurs, therefore even by increasing the budget we do not have any further conversion. This issue is partially overcome with the use of a more fine-grained targeting provided by the campaign $\mathcal{C}^*$, whose performance are bounded to $\boldsymbol{J}_0(\overline{B}) \approx 12$ for $\overline{B} \geq 260$. We suppose that an even finer-grained targeting, e.g., using 1-hour-long slots, could further improve the performance of the optimal advertising campaign.

B. Evaluation in a Synthetically Generated Setting

Experimental Setting. In this experiment, we compare the performance of the proposed exploration strategies for the A-TargOpt algorithm in a synthetically generated setting. The original campaign $\mathcal{C} = \{s_0\}$ is composed of a single subcampaign s_0 having $K = 3$ features, each of which has cardinality 3. Given an advertising period of $T = 100$ days, we generate for each day $t \in \{1, \dots, T\}$ and for each atomic subcampaign $s_i \subseteq s_0$, the daily observations about the selected bid $b_t(s_i)$, selected budget $p_t(s_i)$, number of clicks $n_t(s_i)$ and cumulative value obtained by the conversions $v_t(s_i)$, in the following way. For each atomic campaign s_i, the policy $\mathfrak{U}_0$ selects a budget $p_t(s_i)$ uniformly over $Y = \{0, \dots, 100\}$, with $|Y| = 10$, and keeps the bid $b_t(s_i) = \bar{c}$ constant during the period and the subcampaigns. This provides an average number of 10 observations for each subcampaign and each budget. For each subcampaign s_i and for each day t, the daily number of clicks is computed as $n_t(s_i) := \frac{p_t(s_i)}{cpc_t(s_i)}$, where the cost per click $cpc_t(s_i)$ is extracted from $\mathcal{N}(0.5, 0.1)$ and $\mathcal{N}(\mu, \sigma)$ is the Gaussian distribution with mean μ and standard deviation σ. We generate the cumulative value obtained by the conversions $v_t(s_i)$ as $v_t(s_i) := cr(s_i)\, n_t(s_i)$ where the conversion rate $cr(s_i)$ for an atomic subcampaign s_i is extracted from $0.5\, B(0.5)$, being $B(\mu)$ the Bernoulli distribution of parameter μ. This cumulative value modeling exemplifies the case in which, on average, half of the atomic subcampaigns is not profitable at all. From the data corresponding to each atomic subcampaign s_i we estimate the lower bound on the number of clicks $\underline{n}(s_i, y)$ as described in Section IV-A. Moreover, the lower bound on the number of clicks $\underline{n}(s_j, y)$ corresponding for the non-atomic subcampaign s_j is computed by aggregating the observations of those atomic subcampaigns s.t. $s_i \subseteq s_j$ and computing the lower bound as in Section IV-A.[5]

In the execution of the A-TargOpt algorithm, we set a total budget to spend of $\overline{B} = 100$ and a number of budget $N_y = 10$. As for performance index, we use the lower bound of the revenue $\boldsymbol{J}_0(\overline{B})$ obtained at the end of the procedure. We compare the performance of our two heuristics with a baseline, called Random Strategy (RS), which expands the node by selecting at random from the leaf nodes. We average the results over 200 independent runs.

Results. The results of the synthetically generated setting are provided in Figure 4. It is possible to see that for $200 \leq N_{\max} \leq 700$ there is statistical evidence that the BFS and OS heuristics are performing better than the RS one. Conversely, with a few nodes ($N_{\max} < 200$) or when the tree is almost completely expanded ($N_{\max} > 700$) there is no significant difference among the performance of the three heuristics. This suggests that our two heuristics provide an advantage when the problem allows the expansion of a considerable portion of the tree, while if we explore only a few nodes, a random exploration might be valid as well.

[5]The aggregation of the data as performed in this experiment is correct under the assumption that all the atomic subcampaigns s_i interacted with the same number of users, thus they contributes to the data of subcampaigns s_j in the same way.

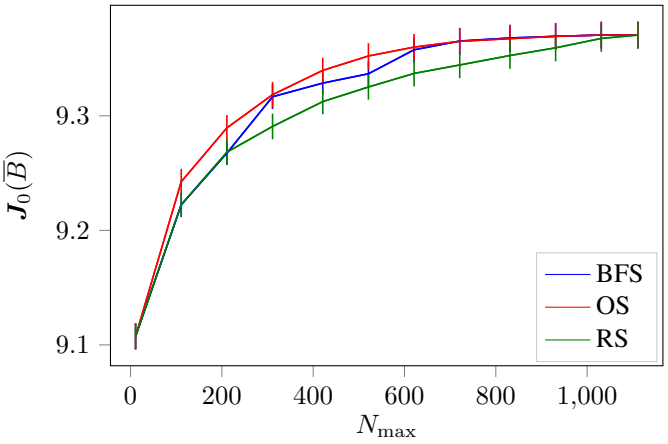

Fig. 4. Revenue obtained by the BFS, OP and RS heuristics by setting a different number of maximum nodes to expand $N_{\max}$. 95% confidence interval are represented as error bars.

In general, looking at the average performance, one should follow the OS heuristic when expanding the nodes, since it is always providing the largest revenue on average. Nonetheless, the heuristic BFS provides solutions which are more compact than OS since the BFS heuristic builds the subcampaigns trying to keep their targets as the large as possible. Therefore, if the marketing experts require a limited number of subcampaigns, e.g., to perform further business analysis on the advertising campaign, the BFS heuristic is to be preferred, while, if we are more concerned about the revenue, we should rely on the OS heuristic when expanding the tree $\mathcal{T}$.

VI. Conclusions

We design a new method to define the optimal target of an advertising campaign, which exploits the information gathered from past interactions between a set of users and the advertising campaign, in the form of logged bandit feedback, to estimate the performance of all the possible subcampaigns in the target space and select the campaign providing maximum revenue. We propose the TargOpt algorithm, which follows the risk-averse framework, to solve the optimization problem to explore completely the target space $\mathcal{M}$. Moreover, in the case the dimension of the target space is too large, we provided the A-TargOpt algorithm, which allows to iteratively expand the space we analyse, and two different heuristics to explore the target space $\mathcal{M}$ effectively, thus providing an anytime version of the TargOpt algorithm. Finally, we showed on both synthetically generated and real-world datasets that the proposed algorithms provide an increase in the revenue gained from the advertising campaign.

An interesting future work is the study of a criterion to decide what is the optimal number of days T after which we run one of the proposed algorithms. Another extension to what has been proposed here is the inclusion of the target optimization procedure in an online learning framework.

Acknowledgments

We would like to thank *Mediamatic*, part of the Marketing Multimedia group, for supporting this research.

References

[1] P. Coopers, "Iab internet advertising revenue report," 2016. [Online]. Available: https://www.iab.com/wp-content/uploads/2016/04/IAB_Internet_Advertising_Revenue_Report_FY_2016.pdf

[2] A. Mas-Colell, M. D. Whinston, J. R. Green *et al.*, *Microeconomic theory*. Oxford university press New York, 1995, vol. 1.

[3] T. Raeder, O. Stitelman, B. Dalessandro, C. Perlich, and F. Provost, "Design principles of massive, robust prediction systems," in *Proceedings of the ACM conference on knowledge discovery and data mining (SIGKDD)*, 2012, pp. 1357–1365.

[4] A. Nuara, F. Trovò, N. Gatti, and M. Restelli, "A combinatorial-bandit algorithm for the online joint bid/budget optimization of pay-per-click advertising campaigns," in *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 2018.

[5] C. Perlich, B. Dalessandro, T. Raeder, O. Stitelman, and F. Provost, "Machine learning for targeted display advertising: Transfer learning in action," *Machine learning*, vol. 95, no. 1, pp. 103–127, 2014.

[6] O. Stitelman, B. Dalessandro, C. Perlich, and F. Provost, "Estimating the effect of online display advertising on browser conversion," *Proceedings of the workshop on Data Mining and Audience Intelligence for Advertising (ADKDD)*, vol. 8, 2011.

[7] F. Provost, B. Dalessandro, R. Hook, X. Zhang, and A. Murray, "Audience selection for on-line brand advertising: privacy-friendly social network targeting," in *Proceedings of the ACM international conference on knowledge discovery and data mining (SIGKDD)*, 2009, pp. 707–716.

[8] J. Yan, N. Liu, G. Wang, W. Zhang, Y. Jiang, and Z. Chen, "How much can behavioral targeting help online advertising?" in *Proceedings of the ACM conference on world wide web (WWW)*, 2009, pp. 261–270.

[9] A. Swaminathan and T. Joachims, "Batch learning from logged bandit feedback through counterfactual risk minimization." *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1731–1755, 2015.

[10] F. Trovò, S. Paladino, P. Simone, M. Restelli, and N. Gatti, "Risk-averse trees for learning from logged bandit feedback," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 976–983.

[11] C. Perlich, B. Dalessandro, R. Hook, O. Stitelman, T. Raeder, and F. Provost, "Bid optimizing and inventory scoring in targeted online advertising," in *Proceedings of the ACM international conference on knowledge discovery and data mining (SIGKDD)*, 2012, pp. 804–812.

[12] S. Thomaidou, K. Liakopoulos, and M. Vazirgiannis, "Toward an integrated framework for automated development and optimization of online advertising campaigns," *Intelligent Data Analysis*, vol. 18, no. 6, pp. 1199–1227, 2014.

[13] J. Xu, K.-c. Lee, W. Li, H. Qi, and Q. Lu, "Smart pacing for effective online ad campaign optimization," in *Proceedings of the ACM international conference on knowledge discovery and data mining (SIGKDD)*, 2015, pp. 2217–2226.

[14] S. C. Geyik, A. Saxena, and A. Dasdan, "Multi-touch attribution based budget allocation in online advertising," in *Proceedings of the workshop on data mining for online advertising (ADKDD)*, 2014, pp. 1–9.

[15] N. Archak, V. Mirrokni, and S. Muthukrishnan, "Budget optimization for online advertising campaigns with carryover effects," in *Proceedings of the ad auctions workshop*, 2010.

[16] W. Zhang, Y. Zhang, B. Gao, Y. Yu, X. Yuan, and T.-Y. Liu, "Joint optimization of bid and budget allocation in sponsored search," in *Proceedings of the ACM international conference on Knowledge discovery and data mining (SIGKDD)*, 2012, pp. 1177–1185.

[17] H. Kellerer, U. Pferschy, and D. Pisinger, *The Multiple-Choice Knapsack Problem*. Springer, 2004, pp. 317–347.

[18] A. Sani, A. Lazaric, and R. Munos, "Risk-aversion in multi-armed bandits," in *Proceedings of the Conference on Neural Information Processing Systems (NIPS)*, 2012, pp. 3275–3283.

[19] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT Press, 2006, vol. 1.

[20] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[21] M. Multimedia, "Mediamatic," 2018. [Online]. Available: http://www.mediamatic.it/