No More Ready-made Deals: Constructive Recommendation for Telco Service Bundling

Paolo Dragone*
University of Trento, Italy
TIM (SKIL), Trento, Italy
paolo.dragone@unitn.it

Giovanni Pellegrini*†
University of Trento, Italy
TIM (SKIL), Trento, Italy
giovanni.pellegrini@unitn.it

Michele Vescovi
TIM (SKIL), Trento, Italy
michele.vescovi@telecomitalia.it

Katya Tentori
University of Trento, Italy
katya.tentori@unitn.it

Andrea Passerini
University of Trento, Italy
andrea.passerini@unitn.it

ABSTRACT

We propose a new recommendation system for service and product bundling in the domain of telecommunication and multimedia. Using this system, users can easily generate a combined service plan that best suits their needs within a vast range of candidates. The system exploits the recent *constructive preference elicitation* framework, which allows us to flexibly model the exponentially large domain of bundle offers as an implicitly defined set of variables and constraints. The user preferences are modeled by a utility function estimated via *coactive learning* interaction, while iteratively generating high-utility recommendations through constraint optimization. In this paper, we detail the structure of our system, as well as the methodology and results of an empirical validation study which involved more than 130 participants. The system turned out to be highly usable with respect to both time and number of interactions, and its outputs were found much more satisfactory than those obtained with standard techniques used in the market.

CCS CONCEPTS

• **Information systems → Personalization**; **Recommender systems**; • **Computing methodologies → Structured outputs**;

KEYWORDS

Constructive Preference Elicitation; Coactive Learning

ACM Reference Format:
Paolo Dragone, Giovanni Pellegrini, Michele Vescovi, Katya Tentori, and Andrea Passerini. 2018. No More Ready-made Deals:, Constructive Recommendation for Telco Service Bundling. In *Twelfth ACM Conference on Recommender Systems (RecSys '18), October 2–7, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240323.3240348

*PD and GP are fellows of TIM-Telecom Italia at the Joint Open Lab SKIL in Trento, and they are both supported by TIM scholarships.
†GP is also supported by the EIT Digital's Doctoral Training Centre (DTC) of Trento.

RecSys '18, October 2–7, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-5901-6/18/10…$15.00
https://doi.org/10.1145/3240323.3240348

1 INTRODUCTION

Product and service bundling has always been a key selling point in the telecommunication industry, as well as in many other sectors [19, 20, 26, 30]. Bundles in the telco market consist in the combination of several services such as mobile connectivity and broadband allocation, usually granted for a monthly payment. These services are often marketed together with other multimedia and entertainment services such as streaming TV, music, and storage service subscriptions. Exploiting the recurrent structure of payments, electronic devices and other expensive products are also often included in these offers to be paid in installments, supporting customer retention. The standard approach used by Telco service providers to market this kind of bundles is to build a handful of pre-made plans to choose from, usually with little opportunity for customization limited to the purchase of some extra feature or service. A multitude of studies reveal, however, that customization through product configuration increases customer satisfaction and brand loyalty [7, 15, 37, 41]. However, configuring a complex product from scratch is not an easy task, and requires a lot of domain expertise from the customer. Recommender systems can substantially contribute to help non-expert users find their preferred solution within the multitude of potential bundles.

The task of recommending personalized bundles of products has been an active research subject over the last couple of decades, addressed by data-driven approaches [2, 24, 48] and especially by constraint-based recommendation systems [45]. Data-driven approaches use association rule mining combined with collaborative information and similarity metrics to create bundles based on frequently purchased item-sets [2, 24, 48]. These methods, however, rely heavily on purchase data and cannot be used to explore the combinatorial space of potential configurations and synthesize fully personalized plans which combine purchase history, personal preferences and company constraints. Constraint-based systems, on the other hand, can easily encode the full domain implicitly as a constraint satisfaction problem [17, 18, 45]. This allows to handle *hard* constraints that the recommended bundles must satisfy, such as compatibility among the items in the bundle, and *soft* constraints that encode user preferences, namely weighted formulas that should be satisfied as much as possible [10, 47]. The function obtained as the weighted sum of soft constraints is the *utility* of the user, which ranks configurations according to the user preferences.

This is the typical setting in which "conversational" recommender systems are employed [44], and the utility function is *learned* via user interaction in a process commonly known as *preference elicitation* [3–6, 9, 33]. Up until recently, however, constraint-based recommenders were lacking a consistent and efficient preference elicitation method, since state-of-the-art preference elicitation techniques [23, 43] could not scale up to large combinatorial spaces without incurring in severe performance issues [14, 39]. This prevented constraint-based systems to really scale up in terms of customization, personalization and usability. This problem was much mitigated with the recent introduction of the *constructive preference elicitation* framework [13].

Constructive preference elicitation combines the benefits of constraint-based recommenders with those of online *structured prediction* algorithms [1, 8]. In this setting, a prominent preference learning framework is coactive learning [36]. Coactive learning is a paradigm for estimating utility functions from weak user feedback, in the form of "improvements" over proposed configurations. In constructive preference elicitation, configurations are generated using a combinatorial optimization solver in which arbitrary domains and constraints can be encoded, in much the same way as it is done in standard constraint-based recommenders. Several variants of this paradigm have been proposed, for handling different kinds of user feedback, such as choice sets [14] and user critiques [38], as well as for scaling up to very large combinatorial spaces [12] and to jointly learn from multiple users [40].

While promising, research work on constructive preference elicitation has remained mainly theoretical with empirical evidence provided only on simplified tasks and user simulations. In this paper, we present a fully functional constructive recommendation system, dubbed *Smart Plan*, for aiding users in finding the bundle of telecom services and products which best fits their preferences. The system was implemented as a web application and tested by more than 130 real participants. Results show that, participants who interacted with our constructive system ended up choosing a satisfactory plan in almost all cases, while this was not true for participants who interacted with a standard system that relied on a fixed number of carefully designed but uneditable plans. Crucially, our results also indicate that a constructive interaction interface, which allows users to request specific changes to recommendations they receive, is useless when it is not backed by a solver capable of synthesizing novel plans.

The rest of the paper is organized as follows. Section 2 overviews the related work, while Section 3 provides a review of coactive learning for constructive preference elicitation. In Section 4 we describe the structure of our system. Our empirical analysis is discussed in Section 5, and conclusions are drawn in Section 6.

2 RELATED WORK

The goal of our system is to generate bundles by exploring a constrained combinatorial space of possible configurations. This is amenable to what is commonly known as a *product configuration* process. Product configuration is a long standing problem in AI [27]. It consists of a step-by-step process in which the user chooses an attribute of the object (which can assume several values) and subsequently chooses the value she prefers for that attribute. The job of a configurator system is to make sure that the chosen values comply with a set of feasibility constraints defined over the attributes. This process is advantageous for configuration tasks in which the user is a domain expert (e.g. manufacturing processes). In case the user is not a domain expert, like for telephone service bundling, the configuration process may result too daunting [42]. Moreover, psychological studies [32] have shown that users adapt their preferences while browsing for solutions, which is difficult to achieve if users are only presented with sequences of choices on single attributes. For this reason, in many cases reasoning in terms of complete, feasible configurations might be a better option than letting the user configure a product from scratch.

Constraint-based systems [17] are used to recommend complete configurations of complex products that must meet certain requirements, both in terms of feasibility constraints and user preferences. Service bundle recommendation has already been explored in the context of constraint-based systems [45], also incorporating user preferences as *soft* constraints [47] and in combination with utility functions to rank feasible configurations [28, 46]. However, these techniques typically lack a systematic way to elicit user feedback and *learn* a utility model from it.

Preference elicitation [33] is a methodology for estimating a utility model of the user preferences while recommending configurations of progressively higher quality. Preference elicitation is an iterative process in which the system makes recommendations to the user, who in turn provides feedback. The most well known preference models are multi-attribute utility functions, which express the utility of a configuration in terms of its components [29]. These are especially convenient to represent user preferences over combinatorial configuration spaces, as in the case of configurable products and bundles. The state-of-the-art techniques for preference elicitation are based on Bayesian estimation over the parameters of the utility functions [23, 43]. These techniques, however, are very computationally demanding and do not scale to truly combinatorial problems [39]. In order to overcome these limitations, a novel framework called *constructive preference elicitation* [13, 39] has been recently proposed by combining preference elicitation, *structured-output prediction* [1] and combinatorial optimization.

In constructive preference elicitation, the utility model is learned from noisy user feedback using an online structured learning algorithm, while inference is addressed using a combinatorial optimization solver. Constructive algorithms have been devised for handling choice set feedback [14, 39], incorporating critiquing feedback [38], and, most closely related to this work, *coactive feedback* [11, 12, 38].

Coactive learning [36] is an online structured prediction framework for learning utility functions from weak user feedback. Coactive learning is based on well known online learning algorithms [8, 34, 49] and can learn from noisy and imperfect user feedback. The coactive interaction consists in an iterative process in which the algorithm suggests the best object according to its current utility estimate, and the user provides an "improvement" of the recommended object. While coactive learning was primarily conceived for optimizing rankings from implicit feedback [35], in the context of constructive preference elicitation it was adapted to learn over product configurations from a combination of explicit and implicit feedback. Coactive learning has also been extended in several ways, such as learning and making predictions from a combination of

an individual and a global model of the users preferences [22] or speeding up inference by relying on locally optimal solutions without giving up theoretical guarantees [21]. Being based on coactive learning, our system can be readily extended in all these directions.

3 THE LEARNING ALGORITHM

In this section we briefly revise the constructive preference elicitation and coactive learning frameworks, laying down the foundations on top of which our system is built.

The goal of our constructive recommendation system is to learn to generate high-utility objects for the user. The most general formulation of utility considered in this framework is a function $u : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, where $\mathcal{X}$ is a set of *contexts*, while $\mathcal{Y}$ is a space of output objects the user chooses from. The context represents here relevant known information which affects the utility of the outputs, e.g. user profile information when recommending jobs, or content of the fridge when suggesting purchases at the supermarket.

Constructive preference elicitation deals with *structured* objects, i.e. objects formed by several components and subject to several feasibility constraints. This is the typical case in product configuration: objects are products with many components which can be composed in several ways, modulo incompatibilities or other constraints. Natural examples of structured objects handled by constructive systems are configurable products like personal computers and furniture arrangements [11], or even trip plans [38] and training schedules [12]. Telephone plans fit this description too: they are composed of several variables, such as the amount of available GB of data traffic for the mobile connection, the list of mobile app or entertainment services associated with the plan (TV on demand, music streaming, etc.), and the devices sold with the plan (smartphone, tablet, etc.). Feasibility constraints include, for instance, no music streaming app without at least one GB of mobile data traffic.

To recommend an optimal object in this setting, we need to be able to *learn* a good utility model of the user preferences and then *predict* the best structured object $y \in \mathcal{Y}$ for the user in some context $x \in \mathcal{X}$ according to the learned model. We cast this problem into a *structured-output prediction* task [1], in which supervision is acquired through user interaction. A user interacts with the algorithm in an iterative fashion. At each iteration the algorithm suggests an object to the user and the user provides some feedback, which the algorithm uses to update the current utility model. While many types of feedback and update schemas can be employed, our system is based on the interaction schema from the coactive learning framework [36]. Algorithm 1 shows a generic algorithm based on the coactive interaction schema. In coactive learning, throughout the iterations $t \leq T$, the algorithm keeps a utility model $u_t(x, y)$ of the form $\langle \boldsymbol{w}_t, \boldsymbol{\phi}(x, y) \rangle$, where $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ is a function mapping contexts and objects to d-dimensional feature vectors and $\boldsymbol{w}_t \in \mathbb{R}^d$ is a vector of parameters to be learned. The algorithm starts by initializing the weights $\boldsymbol{w}_1$ to a reasonable guess (line 1). This could be achieved by, e.g. training the weights over generic preference data from other users or setting them to some reasonable values given the profiling information of the user (see Section 4.2 for an actual implementation of this initialization mode). At each iteration t, the algorithm receives a context $x_t \in \mathcal{X}$ (line 3), which could be external information available at prediction time

Algorithm 1 A generic coactive learning algorithm [36].

1: Initialize $\boldsymbol{w}_1$
2: **for** $t = 1, \ldots, T$ **do**
3: Receive user context x_t
4: $y_t \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}(x_t)} \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y) \rangle$
5: Receive improvement $\bar{y}_t$ from the user
6: $\boldsymbol{w}_{t+1} \leftarrow \Pi_{\mathcal{B}} \left(\boldsymbol{w}_t - \eta_t \left(\boldsymbol{\phi}(x_t, y_t) - \boldsymbol{\phi}(x_t, \bar{y}_t) \right) \right)$

(e.g. the content of the fridge) or some user input (e.g. the type of dish to be prepared). This is the most general type of context, and accounts for simpler cases like context-independent utility ($x_t = \varnothing$ for all t) and fixed context ($x_t = x_1$ for all t).

Based on the current utility estimate u_t and on the context x_t the algorithm "predicts" an object y_t by maximizing the utility $u_t(x_t, y)$ over the space $\mathcal{Y}(x_t) \subseteq \mathcal{Y}$ (line 4). In the case of generic structured objects, composed of both integer and continuous variables and subject to arbitrary constraints, this inference procedure is carried out by casting the optimization problem into a mixed integer program (MIP). By restricting the features $\boldsymbol{\phi}$ and the feasibility constraints over $\mathcal{Y}(x_t)$ to be linear in the variables of y the problem becomes a mixed integer linear program (MILP). This type of problems are well studied in the literature and off-the-shelf solvers can scale to problems with several hundred variables.

The predicted object y_t is recommended to the user. In turn the user replies with feedback in the form of an "improvement" over y_t (line 5), i.e. a new object $\bar{y}_t \in \mathcal{Y}$ that is (even slightly) preferred to y_t by the user. Note that it is not required to get explicit improvements from the users but instead it is possible to extract the improved object $\bar{y}_t$ from implicit user feedback [36]. While working with implicit feedback makes the improvement noisier, coactive algorithms are still able to learn accurate models, which makes this framework very flexible.

In the context of preference elicitation for configurable objects, we usually get explicit feedback, which may be either direct or indirect. The former is collected by letting the user directly modify a component of the object (e.g. by adding a service to the phone plan); the latter, instead, is given by an indirect change to some feature of the object (e.g. by telling the system to decrease the price). This second case is the one used in our system (we provide more details in Section 4). After receiving the improvement $\bar{y}_t$, the system can use the ranking pair $(y_t, \bar{y}_t)$ to update the current model. The algorithm takes a gradient step in the direction minimizing the ranking loss (i.e. the difference in utility between recommended and improved objects), with learning rate $\eta_t \in \mathbb{R}$ (typically decreasing with t). In order to regularize the learned model, the updated weight vector can then be projected onto a convex set $\mathcal{B} \subseteq \mathbb{R}^d$ using a projection operator $\Pi_{\mathcal{B}}(\theta) = \operatorname{argmin}_{w \in \mathcal{B}} \|w - \theta\|$. The set $\mathcal{B}$ is typically a ℓ_2 d-dimensional ball of a given radius, or alternatively an ℓ_1 ball can be used to encourage sparsification of the weights [16].

For simplicity of exposition, the algorithm terminates after a predefined number of iterations T. In a real setting, the user decides when to stop the interaction according to her satisfaction with the recommended configuration. Accordingly, this is the criterion we used in our experiments with real users.

Mobile Connectivity	
Gigabytes	`[0, 2, 4, 6, . . . ]`
Minutes	`[0, 500, 1000, 1500, 2000, ∞]`
Landline	
Internet landline	`[None, ADSL, Broadband]`
Phone landline	`[None, Pay-per-minute, Unlimited]`
Multimedia	
TV on demand	`[Netflix, Infinity TV, . . . ]`
Sky	`[Sky Tv, Sky Sport, . . . ]`
Music	`[Spotify, Apple Music, . . . ]`
Apps & Games	`[Audible, Playstation Plus, . . . ]`
Devices	
Smartphones	`[iPhone 8, Samsung Galaxy S8, . . . ]`
Tablet	`[iPad Pro 10, Galaxy Tab S2, . . . ]`
TV	`[Samsung TV 28", 4K 43", . . . ]`
Laptops	`[Macbook Air 13", . . . ]`

Table 1: Summary of the structure of the recommended telephone plans. Left column: component names. Right column: outline of the possible values for each component.

4 THE SMART PLAN SYSTEM

In this section we describe the Smart Plan system, a constructive recommender system for integrated telephone plans based on coactive learning. The system recommends configurations $y \in \mathcal{Y}$ formed of several components. The components are organized into four groups: [i] Mobile Connectivity services; [ii] Landline services; [iii] Multimedia (apps and services for entertainment and multimedia content provisioning such as TV streaming, music, etc.); [iv] Devices (such as smartphones, tablets, etc. paid in monthly installments). Table 1 summarizes the groups and their components, outlining the range of possible values for each component. In total, there are 12 basic components, each taking on average about 5 possible values, which combined form a number of possible configurations in the order of $5^{12} \approx 10^9$. The feasible space $\mathcal{Y}$ is a subset of the full set of combinations that is determined by hard constraints over the attributes. For example a landline internet connection is needed to get the pay TV service, or a mobile subscription (either voice minutes or GB of data traffic) is needed to include a smartphone in the plan. Besides the basic components, the plans also contain several derived attributes, such as the total price of the plan (computed as a function of the included components), or the amount of monthly payments due for the installments of the devices included in the plan. At the beginning of the interaction, the system asks the user to select one out of four *categories* of plans. The categories are: [i] "Young" (plans for users under 30); [ii] "Family" (plans for families with children); [iii] "Business" (high-end plans for business); [iv] "Flex" (for everybody else). The user choice of category becomes the context $x \in \mathcal{X}$ for the full elicitation process[1]. The category x may further alter the feasible space $\mathcal{Y}(x) \subseteq \mathcal{Y}$ of plans, e.g. prices of certain services are lower for the "Young" category.

The feature vector $\boldsymbol{\phi}$ includes about 160 features describing the components of the plan and their interconnections, for instance several features describing different ranges of minutes amount (e.g. *minutes* $\leq \{200, 500, 1000, \dots\}$), or a feature encoding the difference between the final price and the price payed after the installments due for the devices. A complete description of the components, constraints and features is given in the supplementary material.

The system is implemented as a web application composed of: [i] a web interface with which users interact; [ii] a web service connected to a learning back-end which is in charge of generating recommendations and collecting data.

4.1 The user interface

The user interface of the system is a web page alike to that shown in Figure 1. The page shows one recommended plan at the time, displayed as a grid containing all of its components. The grid separates the various groups of components and arranges the various components within the groups, showing placeholders for components that are not present in the given plan. For each group, the sub-total price is displayed. For the device group, the amount of monthly payments due for the devices included in the plan is shown as well. The total price of the plan is reported on the right side of the grid. A discount is sometimes applied to the total price depending on the number and type of services in the configuration. Additional information on the different components is displayed using tooltips and overlays on mouse over. The user can interact with the system via the buttons shown in the right part of the grid. At any given time the user can: [i] choose the currently displayed plan; [ii] suggest some changes to the current plan and request an new one; [iii] exit the session without choosing any plan. The system starts with an initial recommendation depending on the category chosen at the beginning and then computes new recommendations every time the user suggests changes. The plans in the history are numbered and the user can navigate the entire history of recommendations using the appropriate buttons (top right), and focus on any of them (and not only the last one) for the interaction. In any case, novel recommendations will be enqueued as the last plan in the history.

The right screenshot in Figure 1 shows the interface once the "Suggest changes" button has been pressed. In this state, several toggle buttons appear underneath the components that can be changed. In the case of numerical values and ordinal values, such as the amount of gigabytes for the mobile connection or the proposed monthly price of the plan, the user can suggest to increase or decrease the value, or ask it to be left unchanged. For categorical components, such as the multimedia services and devices, the user can suggest to add a service if it is not present, or, if it is present, the user can suggest to remove it, change it or keep it as it is. As the system needs to make trade-offs between the current configuration, the constraints of the domain, the current preference model and the user feedback, the user suggestion of changing one component may result in a change of other components as well. For instance, if the user asks to add more gigabytes to the plan and asks for a lower price, the system will need to remove some of the other services to accommodate this opposite feedback. This is the reason why we also included the "equal" button, with which the user can suggest which components should not change (e.g. because they are considered important) when computing these trade-offs. In general, however, we warn the user (via an overlay) to keep the number

[1]Henceforth, the terms "category" and "context" are used interchangeably.

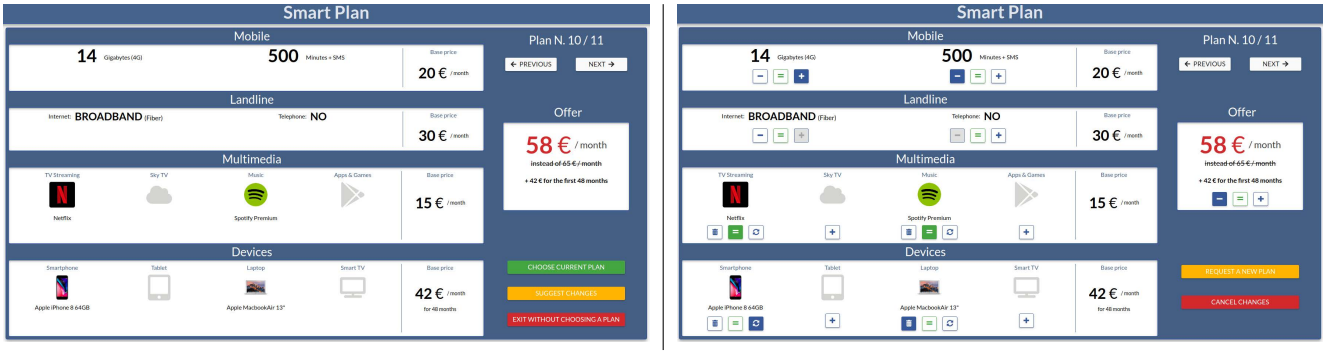

Figure 1: Two screenshots of the web interface of our system. On the left, a screenshot of the view of a recommended plan while navigating throughout the recommendation history. On the right, a screenshot showing the interface the user can interact with when selecting the changes to suggest to the system. (Best viewed in color.)

of suggestions per iteration as low as possible in order to increase the chance of having them satisfied, reminding her that there is no limit to the number of plans she can require from the system.

4.2 The learning subsystem

This component of the system is in charge of providing recommendations, keeping track of the user feedback and updating the user utility model accordingly. This subsystem is further divided into the actual learning algorithm, and a web service API, that intermediates between the constraint solver used to infer new recommendations, a database storing the collected data and the user interface. Here we outline the learning algorithm used in the Smart Plan system, while a more in depth description of the full system implementation is given in the supplementary material.

The learning algorithm used in our system (Algorithm 2) is a slightly adapted version of the coactive learning one (Algorithm 1) to accommodate the type of feedback received by our user interface.

At the beginning the algorithm receives the category $x \in \mathcal{X}$ of user choice. Based on the category x, the algorithm sets the initial weights $\boldsymbol{w}_0 = \boldsymbol{w}_x$ and the initial recommendation $y_0 = y_x$. Selecting a meaningful starting point for both the initial configuration and the initial weight vector can drastically speed-up the elicitation process. In principle, initial configurations may be estimated from generic data of previous users if available, using e.g. collaborative or content-based techniques. We did not have previous data on this task, so, with the help of a domain expert, we identified four realistic initial configurations $y_x \in \mathcal{Y}(x)$, one for each category x. The four initial configurations attempt to address the need of a prototypical user who chooses a certain category. For instance, a user choosing the category "young" would probably need more mobile data traffic than a user choosing the "family" one, probably more interested in more voice minutes and landline services (more details on the initial plans y_x can be found in the supplementary material). The initial weights $\boldsymbol{w}_x$ are set to $\text{argmax}_{\boldsymbol{w} \in \mathcal{B}} \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y_x) \rangle$, i.e. the value for which y_x is the highest scoring configuration for the category x. The algorithm also initializes a list $\mathcal{H}$ where the plans recommended at each iteration will be stored.

At each iteration, the algorithm first presents to the user the plan $y_t = \bar{y}_{t-1}$ generated at the end of the previous iteration (which, except for the first iteration, is the one trying to meet user feedback on the previous recommendation) and stores it in the list $\mathcal{H}$. If the user is unsatisfied by the current recommendation (as well as all previous ones) and decides to quit the interaction, the algorithm stops without recommending any plan. Otherwise, the user chooses a plan from the list $\mathcal{H}$ and can either accept it (the algorithm stops and returns it as the final recommendation) or suggest some changes. In the latter case, the algorithm updates its weight vector based on the plan selected and the current history. Indeed, the user selection of some y_k from the recommendation history as the plan on which to provide new suggestions is already an implicit source of feedback. Intuitively, if the user chose to improve the last suggested plan (i.e. $k = t$), this implicitly tells us that the last performed improvement was most likely going in the right direction, and thus we can safely update the weights with the ranking pair (y_{t-1}, y_t) (same as in Algorithm 1 but delayed of one iteration). If, instead, $k < t$, it means that the user preferred to "start over" from some previous recommendation y_k, probably because the recommendations y_i, for $k < i \leq t$, turned out to be farther out from the user desiderata than y_k. This implicit feedback provides the ranking pairs (y_i, y_k) for $k < i \leq t$, exploited to update the weights (line 13). Preliminary experiments showed that this is a better choice than just adding the single pair (y_t, y_k).

Finally, the user provides feedback on the chosen plan y_k, and a refined plan accounting for such a feedback is generated. As mentioned in Section 3, the user feedback is not a direct manipulation of the recommended plan but rather a set of "suggestions" on the components that need to be changed or should be kept equal, leaving the system the possibility of adjusting the remaining components to accommodate these suggestions. The improvement $\bar{y}_t$ is then obtained by finding a feasible plan satisfying the suggestions of the user as much as possible. In order to formalize this search problem, plans are encoded in a form that allows to test the satisfaction of the different types of suggestions the user can provide. A plan y_k is thus represented in terms of two component vectors, $\boldsymbol{\varphi}_n(y_k)$ and a $\boldsymbol{\varphi}_c(y_k)$ [2]. The first contains quantitative information for numerical

[2]Note that these vectors differ from the feature vector $\boldsymbol{\phi}$ on which the utility function is defined, although they largely overlap in practice. See the supplementary material for details.

Algorithm 2 The learning algorithm of the Smart Plan system.

1: User selects category $x \in \mathcal{X}$
2: Initialize $\boldsymbol{w}_0$ and $y_0 = \bar{y}_0$ according to x
3: $\mathcal{H} \leftarrow$ empty list
4: **for** $t = 1, 2, \ldots$ **do**
5: Recommend $y_t = \bar{y}_{t-1}$
6: Append y_t into $\mathcal{H}$
7: **if** User quits the interaction **then**
8: **return** $\varnothing$
9: User selects plan $y_k \in \mathcal{H}, k \leq t$
10: **if** User accepts y_k **then**
11: **return** y_k
12: **for** $i = k-1, \ldots, t$ **do**
13: $\boldsymbol{w}_t \leftarrow \Pi_{\mathcal{B}}\left(\boldsymbol{w}_{t-1} - \eta_t \left(\boldsymbol{\phi}(x, y_i) - \boldsymbol{\phi}(x, y_k)\right)\right)$
14: Receive feedback $y_t^=, y_t^{\neq}, y_t^{\pm}$ on plan y_k
15: $\bar{y}_t \leftarrow \textsc{improve}(x, \boldsymbol{w}_t, y_k, y_t^=, y_t^{\neq}, y_t^{\pm})$

or ordinal attributes (e.g. number of minutes or gigabytes) and presence/absence information for categorical ones (e.g. TV Streaming, Laptop). The second contains the one-hot encoding of the categorical attributes (e.g. which TV Streaming among the different options available) [3].

The user suggestions are gathered into three feedback vectors $\boldsymbol{y}_t^{\pm}, \boldsymbol{y}_t^{=}, \boldsymbol{y}_t^{\neq}$. The first feedback vector $\boldsymbol{y}_t^{\pm}$ has the same size as $\boldsymbol{\varphi}_n(y_k)$, and contains 1 and -1 for components the user has requested to respectively increase (by pressing the "plus" button) and decrease (by pressing the "minus" button) in $\bar{y}_t$ with respect to y_k, and 0 otherwise. The second vector $\boldsymbol{y}_t^{=}$ has again the same size as $\boldsymbol{\varphi}_n(y_k)$, and contains for each component the value 1 if the user requested to keep the value unchanged in $\bar{y}_t$ with respect to y_k (by pressing the "equal" button), and 0 otherwise. The third vector $\boldsymbol{y}_t^{\neq}$ has instead the same size as $\boldsymbol{\varphi}_c(y_k)$, and contains 1 on components for which the user has requested to be changed in value (by pressing the "change" button) in $\bar{y}_t$ with respect to their value assumed in y_k. In other words, if the $\boldsymbol{y}_t^{\neq}$ contains 1 for some component, then the new plan $\bar{y}_t$ should contain a different value (not specified which) for that component with respect to y_k.

The improvement $\bar{y}_t$ is obtained by solving an optimization problem that attempts to maximize a "feedback" score $G_t(\bar{y}_t)$, i.e. a function that encodes how well the user suggestion are satisfied. The new plan however should also be of high utility according to the current utility model u_t and should not be too distant from y_k, in order to make the transition from one plan to the next one as smooth as possible. The IMPROVE function (line 15) seeks a plan $\bar{y}_t$ trading-off these aspects, by solving the following optimization problem:

$$\bar{y}_t = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} G_t(y) + \lambda_1 u_t(x, y) - \lambda_2 \|\boldsymbol{\phi}(x, y) - \boldsymbol{\phi}(x, y_k)\|_1 \quad (1)$$

$$G_t(y) = \lambda_3 \langle \boldsymbol{y}_t^{\pm}, \boldsymbol{\delta}_t^n \rangle - \lambda_4 \langle \boldsymbol{y}_t^{=}, |\boldsymbol{\delta}_t^n| \rangle + \lambda_5 \|\boldsymbol{y}_t^{\neq} \circ \boldsymbol{\delta}_t^c\|_0$$

$$\boldsymbol{\delta}_t^n = \boldsymbol{\varphi}_n(y) - \boldsymbol{\varphi}_n(y_k) \qquad \boldsymbol{\delta}_t^c = \boldsymbol{\varphi}_c(y) - \boldsymbol{\varphi}_c(y_k)$$

[3] The vectors $\boldsymbol{\varphi}_n(y_k)$ and $\boldsymbol{\varphi}_c(y_k)$ also contain some more elaborate features, such as the price range and the smartphone category. More details are provided in the supplementary material.

The above optimization problem maximizes a combination of: [i] the utility of y according to the current model $u_t(x, y)$; [ii] the (negated) ℓ_1 distance between y and the selected plan y_k (in feature space); [iii] the feedback score function $G_t(y)$ computing how much y meets user suggestions. The function $G_t(y)$ is itself a combination of: [i] the amount $\langle \boldsymbol{y}_t^{\pm}, \boldsymbol{\delta}_t^n \rangle$ of changes that go in the same direction with respect to the user suggestion (increase if user asked to increase and viceversa); [ii] the number $\langle \boldsymbol{y}_t^{=}, |\boldsymbol{\delta}_t^n| \rangle$ of components that should remain equal but they have not; [iii] the number $\|\boldsymbol{y}_t^{\neq} \circ \boldsymbol{\delta}_t^c\|_0$ of categorical components that changed according to the user request. In the above formula: $|y|$ denotes the element-wise absolute value; $\circ$ denotes the Hadamard product (element-wise product); and $\|\cdot\|_0$ denotes the ℓ_0 norm (number of non-zero elements). The parameters $\lambda_1, \ldots, \lambda_5$ are real coefficients defining the relative importance of each component of the function being maximized. These coefficients, as well as the learning rate η_t, were chosen manually among a set of predefined values, validating the quality of the recommendations on a pilot set of users (beta testers which are not included among the real users involved in the study described in Section 5). The final implementation used a learning rate $\eta_t = \frac{1}{t^{0.8}}$ with a projection onto a ℓ_2 ball. The learning rate decreases rather quickly because, in this particular setting, feedback given in the early iterations is more important than finer adjustments at later iterations when most of the users already had a rough idea in mind of what the final plan should look like. The parameters $\lambda_1, \ldots, \lambda_5$ were set so that the feedback score, primarily the $\boldsymbol{y}_t^{\pm}$ component, had the highest weight. Furthermore, a slightly higher weight was given to the distance $\|\boldsymbol{\phi}(x, \bar{y}_t) - \boldsymbol{\phi}(x, y_k)\|_1$ with respect to the utility $u_t(x, \bar{y}_t)$, so that not too many components would change between y_k and $\bar{y}_t$, thus ensuring a smooth enough transition between plans.

Regarding the implementation, we used the MiniZinc constraint programming language as modeling platform [31]. The underlying constraint solver used was Gurobi [25]. Concerning timing, preliminary experiments showed that inference and learning steps took on average 0.21 seconds per iteration using Gurobi on a 2.8 GHz Intel Xeon CPU with 8 cores and 32 GiB of RAM. This time magnitude makes the user interaction in the practical implementation comfortable and without delays.

5 EMPIRICAL VALIDATION

This section describes the empirical validation of the system through an experiment with real participants. The supplementary material also contains a batch of preliminary experiments made with simulated users in order to assess the convergence rate of the algorithm when interacting with users of different informativeness, as done in previous works on constructive preference elicitation [11, 12, 38].

In order to evaluate the usefulness of our constructive approach, we compared it with two alternative versions of the system. The first mimics the standard approach of telecommunication companies, which consists of hand-crafting a pool of integrated plans tailored for different categories of users, and let them to choose their preferred plan in the pool. The second employs the very same interaction modality of our constructive approach, but when building the recommendation in response to the suggestions from the user, it is forced to pick one of the preset plans in the pool. The

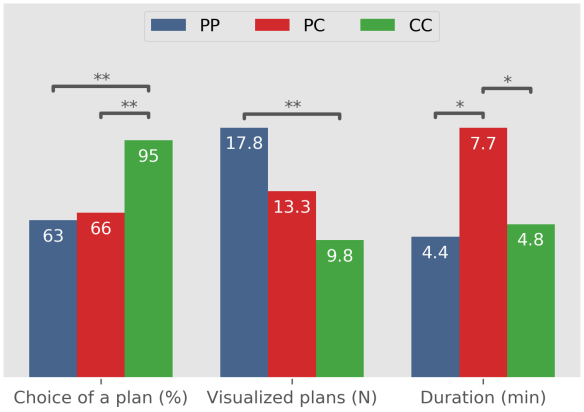

Figure 2: Statistics concerning the interaction with the three (PP, PC, CC) versions of the system. From left to right: [i] the percentage of participants who found a plan satisfying enough to choose it; [ii] the average number of plans visualized (only for participants who chose a plan); [iii] the average duration (in minutes) of the interaction with the system (only for participants who chose a plan). Top bars indicate significant comparisons (*= p<.05 and **= p<.01, adjusted standardized residuals post-hoc analysis for [i], Tukey post-hoc tests for [ii] and [iii]). Best viewed in color.

rationale behind the inclusion of this third version of the system is to evaluate the relative importance of the constructive interface *versus* the constructive plan generation in determining the success of the interaction and the user's satisfaction. In what follows, we refer to our constructive approach as CC (standing for "constructive-constructive"), while the two alternative approaches as PP ("pool-pool") and PC ("pool-constructive"), respectively.

5.1 Stimuli

In the CC version of the system, the recommendations were selected from the full configuration space as described in Sections 4. In the PP version, the recommendations were selected from a predefined pool. The pool was created by reviewing currently available telco companies offers and interacting with a domain expert. This process resulted in 65 different plans, divided into a set of sub-pools $\mathcal{P}_x \subset \mathcal{Y}$ of approximately equal size for each of the "Young", "Family", "Business", and "Flex" categories x (see Section 4). Note that, albeit very small if compared to the size of the configuration space, the pool is still substantially larger than those typically provided by telco companies. Finally, in the PC version, the algorithm learnt a model of the user preferences as for the fully constructive version, but recommendations were selected from the same predefined pool of the PP version. However, while in PC version the full pool was used for inference, in the PP version only the sub-pool associated to the chosen category was used, to make the choice not too overwhelming for the user.

5.2 Participants

A sample of 157 adults was recruited for the experiment. Participation was anonymous and on a voluntary basis. Those who abandoned the session or did not visualized at least two recommendations (15% of the total) were excluded, leaving us with 134 participants. Fifty-eight percent of them were male, the average age was 32.7 years (SD 10.42).

5.3 Procedure

The experiment was performed online, throughout a web page compatible with any widely used web browser. Participants were explicitly informed that the experiment was only for scientific purposes and that the plans were not associated to any real offer. However, they were asked to behave as if they really had to choose a plan to purchase.

A between-subjects design was employed: participants were randomly assigned to one of three groups, each interacting with a different version (CC, PP or PC) of the recommender. More specifically, 34% were presented with the PP, 33% with the PC, and 33% with the CC version of the system. Participants in all groups had first to select the category of plans they were most interested in. The system used the information about the category to suggest an appropriate initial recommendation y_0 (see Section 4.2). In the CC and PC versions of the system, this information was also used to initialize the weight vector $\boldsymbol{w}_0$, whereas in the PP version of the system the category was used to select the pool $\mathcal{P}_x$ to be presented (in a randomized order) to participants.

Participants could spend as much time as they needed to assess each recommended plan. They could then suggest modifications to the plan (for PC and CC versions only, as outlined in Section 4.1), choose the plan, or decide to conclude the experimental session without choosing any plan (or even to leave the test simply by closing the browser window). As described in Section 4.1, in all the versions of the system, participants were free to navigate the full history of recommendations by going back and forth between plans. (Specific instructions on how to interact with the specific version of the system were provided by means of a video tutorial before the start of the experiment.) Finally, once participants had chosen a plan or decided to leave the experimental session without choosing any, they were presented with a small questionnaire. For all participants, this included questions about how pleasant and tiring the interaction with the system had been on a scale ranging from 0 (= "completely disagree") to 5 (= "completely agree"). Other questions depended on the version of system and on whether the user chose a plan or left without choosing any. All participants in the PC and CC versions were also asked whether they had found the plans proposed by the system of growing interest, while all participants who chose a plan were asked how much they were satisfied with it (again from 0 to 5). At the very end, participants were invited to provide minor demographic information (i.e., their age and gender).

5.4 Results

Participants who interacted with the three (PP, PC, and CC) versions of the system were not significantly different by gender ($\chi^2(2)$=4.334, p=.114), age (one-way ANOVA, $F(2,120)$=.617, p=.541),

	Pleasant interaction	Tiring interaction	Increasing interest	Satisfaction degree
PP	3.79	1.07	–	3.54
PC	2.95	2.40	1.51	3.03
CC	3.44	1.79	3.02	3.34

Table 2: Participants' average answers to the final questions in the three (PP, PC, CC) system versions. Value range from 0 to 5. From left to right: [i] how pleasant the interaction with the system was; [ii] how tiring the interaction with the system was; [iii] (only for participants in the PC and CC versions) whether the plans proposed by the system were of growing interest; [iv] (only for participants who chose a plan) how much the chosen plan was considered satisfying.

and possession of a phone plan ($\chi^2(2)=.628$, $p=.731$). They did not significantly differ also in their choice of the category of the plan ($\chi^2(6)=3.763$, $p=.709$). As a consequence, they can be considered fully suitable to be compared in the experiment. Overall, the categories "flex", "young", 'family", and "business" were chosen by 31%, 45%, 18%, and 6% of the participants, respectively.

As shown in Figure 2, the percentage of participants who ended their interaction with the system successfully by choosing a plan was significantly different in the three groups ($\chi^2(2)=15.106$, $p<.001$). In particular, while almost all participants (95%) who interacted with the CC version of the system found a plan that they liked, the same held only for a bit more than half of those who interacted with the other two versions (63% and 66% for PP and PC, respectively). The difference between the former and the latter is statistically significant (adjusted standardized residuals post-hoc analysis, $p<.01$).

Before making their choice, participants in the three groups visualized a different number of plans (one-way ANOVA, $F(2,97)=6.474$, $p<.01$). In particular, participants who interacted with the PP version of the system visualized a greater number of plans than participants who interacted with the CC version (Tukey post-hoc test, $p<.01$). This is not entirely surprising since participants who interacted with the PP version of the system could not suggest themselves a new plan, but had only the possibility to explore the available plans.

Participants in the three groups who chose a plan also differed with regards to the duration of their overall interaction with the system (one-way ANOVA, $F(2,97)=4.622$, $p<.05$). This was greater for participants who interacted with the PC version of the system than participants who interacted with the PP and CC versions (Tukey post-hoc tests, $p<.05$), while participants in the latter two groups did not differ between each other (Tukey post-hoc test, $p=.935$). Such a result indicates that the PC version of the system was especially time-consuming, while the CC version was comparable to the PP one.

The average answers to the final questions are reported in Table 2. Unsurprisingly, the three versions of the system have not been rated as equally enjoyable or tiring (one-way ANOVAs, $F(2,126)=5.928$, $p<.01$, and $F(2,126)=8.703$, $p<.01$, respectively). More specifically, Tuckey post-hoc tests revealed that interacting with the PC version of the system was considered as less pleasant (<.01) and more tiring (p<.01) than interacting with the PP version, while there were no significant differences between PC and CC versions ($p=.33$ and $p=.06$, respectively). This result indicates that combining a constructive interface with a standard search over a fixed pool of candidates is not a good strategy, since a restricted number of options cannot typically accommodate the modifications suggested by the users. Moreover, it shows that the interaction with the CC version of the system, although more engaging, is not perceived as less pleasant or more tiring than the interaction with the PP one.

Participants who interacted with the CC version of the system found the plans progressively suggested to them of growing interest more than they did participants who interacted with the PC version (Independent t-test, $t(84)=5.972$, p<.01). Yet again, this suggests that participants' appreciation of the interaction depends on having a constructive plan generator rather than a constructive interface.

Finally, there were not significant differences in the satisfaction with the chosen plan between the three groups ($F(2,95)=1.911$, $p=.154$).

6 CONCLUSION

In this paper, we presented Smart Plan, a novel type of recommender system for product and service bundling in the telecommunication and multimedia domain. Rather than presenting the user a fixed set of carefully designed but uneditable plans to choose from, our system leverages the constructive preference elicitation framework with coactive learning to synthesize progressively better plans while interacting with the user. An empirical validation with real participants showed that when interacting with Smart Plan, participants ended up choosing a plan almost in all cases, much more often than when having to select from a fixed pool of candidates, regardless of the interface used to explore the pool. Interestingly, the empirical results also show that the satisfaction degree does not differ significantly across different systems. This might be highlight either a limitation of the method in providing better recommendations than the others (even if providing satisfactory recommendation *more often*), or a stronger correlation of the satisfaction degree with the plan itself rather than the process for acquiring it. A more in depth analysis is required to better assess the impact of the recommendation process on the satisfaction degree.

To the best of our knowledge, this is the first system of its kind and could contribute to a paradigm shift in the way telco offers are marketed to the public. Furthermore, the machine learning methodology underlying our system can be straightforwardly applied to many other domains, such as finance, banking, or insurance, just to mention a few. The chances for improvement are endless. In terms of user interaction, additional forms of user feedback can be introduced, such as choice set [14] or critiquing [38]. In terms of modeling, components of the plans, features and initial plans, which are currently pre-determined according to domain expertise, can be adapted based on customers' usage trends. Note that this can be seen as a form of *domain learning*, where the analysis of previous customers' behavior not only allows to provide more appropriate suggestions to new ones (e.g., better initial plans), but can also guide the refinement of the space of feasible configurations.

REFERENCES

[1] Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. 2007. *Predicting Structured Data.* MIT Press.

[2] Moran Beladev, Lior Rokach, and Bracha Shapira. 2016. Recommender systems for product bundling. *Knowledge-Based Systems* 111 (2016), 193–206.

[3] Craig Boutilier. 2002. A POMDP formulation of preference elicitation problems.

[4] Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. 2004. Preference-Based Constrained Optimization with CP-Nets. *Computational Intelligence* 20, 2 (2004), 137–157.

[5] Craig Boutilier, Ronen I Brafman, Holger H Hoos, and David Poole. 1999. Reasoning with conditional ceteris paribus preference statements. In *UAI*. 71–80.

[6] Darius Braziunas. 2006. *Computational Approaches to Preference Elicitation.* Technical Report. Department of Computer Science, University of Toronto.

[7] Pedro S Coelho and Jörg Henseler. 2012. Creating customer loyalty through service customization. *European Journal of Marketing* 46, 3/4 (2012), 331–356.

[8] Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 111.

[9] Carmel Domshlak, Eyke Hüllermeier, Souhila Kaci, and Henri Prade. 2011. Preferences in AI: An overview. *Artificial Intelligence* 175, 7-8 (2011), 1037–1052.

[10] Carmel Domshlak, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. 2003. Reasoning about soft constraints and conditional preferences: complexity results and approximation techniques. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. 215–220.

[11] Paolo Dragone, Luca Erculiani, Maria Teresa Chietera, Stefano Teso, and Andrea Passerini. 2016. Constructive Layout Synthesis via Coactive Learning. In *Constructive Machine Learning workshop, NIPS*.

[12] Paolo Dragone, Stefano Teso, Mohit Kumar, and Andrea Passerini. 2018. Decomposition Strategies for Constructive Preference Elicitation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

[13] Paolo Dragone, Stefano Teso, and Andrea Passerini. 2017. Constructive Preference Elicitation. *Frontiers in Robotics and AI* 4 (2017), 71.

[14] Paolo Dragone, Stefano Teso, and Andrea Passerini. 2018. Constructive Preference Elicitation over Hybrid Combinatorial Spaces. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

[15] Xuehong Du, Jianxin Jiao, and Mitchell M Tseng. 2006. Understanding customer satisfaction in product customization. *The International Journal of Advanced Manufacturing Technology* 31, 3-4 (2006), 396–406.

[16] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*. ACM, 272–279.

[17] Alexander Felfernig and Robin Burke. 2008. Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th international conference on Electronic commerce*. ACM, 3.

[18] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. 2015. Constraint-based recommender systems. In *Recommender Systems Handbook*. Springer, 161–190.

[19] Joshua S Gans and Stephen P King. 2006. Paying for loyalty: Product bundling in oligopoly. *The Journal of Industrial Economics* 54, 1 (2006), 43–62.

[20] Begoña García-Mariñoso and Xavier Martínez Giralt. 2008. Bundling in telecommunications. (2008).

[21] Robby Goetschalckx, Alan Fern, and Prasad Tadepalli. 2014. Coactive Learning for Locally Optimal Problem Solving.. In *AAAI*. 1824–1830.

[22] Robby Goetschalckx, Alan Fern, and Prasad Tadepalli. 2015. Multitask Coactive Learning.. In *IJCAI*, Vol. 15. 3518–3524.

[23] Shengbo Guo and Scott Sanner. 2010. Real-time multiattribute Bayesian preference elicitation with pairwise comparison queries. In *AISTATS*. 289–296.

[24] Liu Guo-rong and Zhang Xi-zheng. 2006. Collaborative filtering based recommendation system for product bundling. In *Management Science and Engineering, 2006. ICMSE'06. 2006 International Conference on*. IEEE, 251–254.

[25] Inc. Gurobi Optimization. 2016. Gurobi Optimizer Reference Manual. http://www.gurobi.com

[26] Andreas Herrmann, Frank Huber, and Robin Higie Coulter. 1997. Product and service bundling decisions and their effects on purchase intention. *Pricing Strategy and Practice* 5, 3 (1997), 99–107.

[27] Lothar Hotz, Alexander Felfernig, Markus Stumptner, Anna Ryabokon, Claire Bagley, and Katharina Wolter. 2014. Configuration knowledge representation and reasoning. (2014).

[28] Dietmar Jannach, Markus Zanker, and Matthias Fuchs. 2009. Constraint-based recommendation in tourism: A multiperspective case study. *Information Technology & Tourism* 11, 2 (2009), 139–155.

[29] Ralph L Keeney and Howard Raiffa. 1993. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge university press.

[30] Jan Krämer. 2009. Bundling Telecommunications Services: Competitive Strategies for Converging Markets. (2009).

[31] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. 2007. MiniZinc: Towards a standard CP modelling language. In *CP*. 529–543.

[32] John W Payne, James R Bettman, and Eric J Johnson. 1993. *The adaptive decision maker*. Cambridge University Press.

[33] Gabriella Pigozzi, Alexis Tsoukiàs, and Paolo Viappiani. 2016. Preferences in artificial intelligence. *Ann. Math. Artif. Intell.* 77, 3-4 (2016), 361–401.

[34] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. 2007. (Online) Subgradient Methods for Structured Prediction. In *Artificial Intelligence and Statistics*. 380–387.

[35] Pannaga Shivaswamy and Thorsten Joachims. 2011. Online Learning with Preference Feedback. In *NIPS Workshop on Choice Models and Preference Learning*.

[36] Pannaga Shivaswamy and Thorsten Joachims. 2015. Coactive Learning. *JAIR* 53 (2015), 1–40.

[37] Ching-I Teng. 2010. Customization, immersion satisfaction, and online gamer loyalty. *Computers in Human Behavior* 26, 6 (2010), 1547–1554.

[38] Stefano Teso, Paolo Dragone, and Andrea Passerini. 2017. Coactive Critiquing: Elicitation of Preferences and Features. In *AAAI*.

[39] Stefano Teso, Andrea Passerini, and Paolo Viappiani. 2016. Constructive Preference Elicitation by Setwise Max-Margin Learning. In *IJCAI*. 2067–2073.

[40] Stefano Teso, Andrea Passerini, and Paolo Viappiani. 2017. Constructive Preference Elicitation for Multiple Users with Setwise Max-margin. In *International Conference on Algorithmic DecisionTheory*. Springer, 3–17.

[41] Sriram Thirumalai and Kingshuk K Sinha. 2011. Customization of the online purchase process in electronic retailing and customer satisfaction: An online field study. *Journal of Operations Management* 29, 5 (2011), 477–487.

[42] Alessio Trentin, Elisa Perin, and Cipriano Forza. 2013. Sales configurator capabilities to avoid the product variety paradox: Construct development and validation. *Computers in Industry* 64, 4 (2013), 436–447.

[43] Paolo Viappiani and Craig Boutilier. 2010. Optimal Bayesian recommendation sets and myopically optimal choice query sets. In *NIPS*. 2352–2360.

[44] Paolo Viappiani, Pearl Pu, and Boi Faltings. 2007. Conversational recommenders with adaptive suggestions. In *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 89–96.

[45] Markus Zanker, Markus Aschinger, and Markus Jessenitschnig. 2010. Constraint-based personalised configuring of product and service bundles. *International Journal of Mass Customisation* 3, 4 (2010), 407–425.

[46] Markus Zanker and Markus Jessenitschnig. 2009. Case-studies on exploiting explicit customer requirements in recommender systems. *User Modeling and User-Adapted Interaction* 19, 1-2 (2009), 133–166.

[47] Markus Zanker, Markus Jessenitschnig, and Wolfgang Schmid. 2010. Preference reasoning with soft constraints in constraint-based recommender systems. *Constraints* 15, 4 (2010), 574–595.

[48] Tao Zhu, Patrick Harrington, Junjun Li, and Lei Tang. 2014. Bundle recommendation in ecommerce. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 657–666.

[49] Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 928–936.